Automatic Schedule Computation for Distributed Real-Time Systems using Timed Automata

Young Saeng Park

A thesis submitted in partial fulfilment
of the requirements of the
University of Northumbria at Newcastle
for the degree of
Doctor of Philosophy

Research undertaken in the School of Computing, Engineering &
Information Sciences

March 2007

Contents

1 Introduction

1.1 Background 1
1.1.1 Real-Time Systems 1
1.1.2 Scheduling Difficulty in Distributed Real-Time Systems 3
1.1.3 Timed Automata 5
1.2 The Central Proposition 6
1.3 Summary of Contributions 8
1.4 Organisation of the Thesis 8

2 Background of Real-Time Scheduling Theory

2.1 Real-Time Scheduling Principle 10
2.1.1 Types of Tasks 11
2.1.2 Classification of Scheduling 12
2.1.2.1 Static Cyclic Scheduling 12
2.1.2.2 Fixed Priority Scheduling 14
2.1.2.3 Dynamic Priority Scheduling 16
2.1.2.4 Comparison between Fixed Priority and Static Cyclic Scheduling 17
2.1.3 Schedulability Analysis 18
2.1.3.1 Processor Utilization Analysis 18
2.1.3.2 Response Time Analysis 21
2.1.3.3 Schedulability Analysis in Distributed Real-Time Systems 23
2.2 Event-Triggered and Time-Triggered Architecture 24
2.2.1 Predictability 25
2.2.2 Resource Utilization 25
2.2.3 Extensibility 26
2.2.4 Testability 27

2.3 Global Scheduling 27
2.3.1 Heuristic Approach 28
2.3.1.1 List Scheduling 28
2.3.1.2 Iterative Deepening A* 29
2.3.2 Simulated Annealing 31
2.3.3 Genetic Algorithm 32
2.3.4 Tabu Search 34
2.4 Summary 35

3 Background of Timed Automata
3.1 Timed Automata 37
3.1.1 Clocks 38
3.1.2 Clock Constraints 39
3.1.3 Timed Automaton 39
3.1.4 Composition of Timed Automata 40
3.1.5 State-Transition System 40
3.1.6 An Example of a Timed Automata Model 41
3.2 Techniques for the Verification of Timed Automata 42
3.2.1 Clock Regions 42
3.2.2 Clock Zones 44
3.3 The Application of Timed Automata to Schedulability 45
3.4 Summary 49

4 Scheduling with Timed Automata
4.1 Formal Definitions of the Terms 50
4.1.1 System 51
4.1.2 Task and Message 51
4.1.3 Precedence Constraint 52
4.1.4 Job 53
4.1.5 An Example of Formal Expression 53
4.2 Design Principle of Schedules with Timed Automata 54
4.2.1 A Timed Automaton for a Task and Message 55
4.2.2 A Timed Automaton for a Job 55
4.2.3 A Timed Automaton for a System Schedule 57
4.2.4 An Example of a Timed Automata model for a Job 58

4.3 Feasible Schedules from Timed Automata 61
4.3.1 Feasible Schedules 61
4.3.2 Finding Feasible Schedules 62
4.4 Summary 64

5 Efficient Timed Automata Models for Scheduling
5.1 Overview of UPPAAL 65
5.1.1 UPPAAL 65
5.1.2 Syntax 66
5.1.3 An Example of a UPPAAL model: Olympic Boxing Scoring System 67
5.1.3.1 Description 67
5.1.3.2 Modelling in UPPAAL 68
5.1.4 Verification 71
5.1.5 Model-Checking and State Explosion Problem 72
5.2 Scheduling Models with UPPAAL 73
5.2.1 Model One 73
5.2.2 Model Two 75
5.2.3 Model Three 76
5.3 Evaluation of the Models 78
5.4 The Limitation of the Timed Automata Model 82
5.5 Summary 84

6 Scheduling Generation with UPPAAL Models
6.1 The Proposed Approach 85
6.2 Prototype Toolset Based on the Approach 89
6.2.1 Fluid Control System 89
6.2.2 Preprocessor 91
6.2.2.1 Description of a System with XML 91
6.2.2.2 Description of a Processor with XML 92
6.2.2.3 Description of a Network with XML 93
6.2.2.4 Description of a Job with XML 95
6.2.2.5 Description of the FC System with XML in the Tool 96
6.2.3 Analyser Engine 96
6.2.3.1 Timed Automata Generator 97
6.2.3.2 Trace Analyser 101

6.2.3.3 Graphic Viewer 102
6.2.3.4 Schedule Extractor 104
6.3 Comparison of the Proposed Approach 105
6.4 Summary 108

7 Case Studies

7.1 Case Study: Adaptive Cruise Control System 110
7.1.1 Description 110
7.1.2 Structure 111
7.1.3 Operation 112
7.1.4 System Description with XML 115
7.1.5 Outputs from the Prototype Toolset 117
7.1.6 Schedule for the ACC System 120
7.2 Case Study: Robot Transport System 121
7.2.1 Description 122
7.2.2 Structure 123
7.2.3 Operation 125
7.2.4 Description with XML 130
7.2.5 Experimental Studies 132
7.2.5.1 Influence of Arbitrary Time Units – Study A 132
7.2.5.2 Influence of Appearance of Jobs – Study B 134
7.2.5.3 Influence of Utilisation of Processors and Networks – Study C 137
7.2.5.4 Influence of Different Timing Values – Study D 140
7.3 Summary 142

8 Further Schedule Constraints

8.1 Schedules Considering Jitter 143
8.1.1 Jitter Control with Temporal Properties – Method A 146
8.1.2 Jitter Control with UPPAAL Models – Method B 149
8.1.3 Comparison between using Temporal Properties and UPPAAL Models 151
8.2 Scheduling Compaction 152
8.3 Precedence Constraints between Jobs 159
8.3.1 Precedence Constraint with Any Time – Case ① 161
8.3.2 Precedence Constraint without Time Delay – Case ② 162

8.3.3 Precedence Constraint with Offset Time – Case ③ 162
8.3.4 Experiments with Precedence Constraints 163
8.4 Summary 167

9 Conclusion and Future Work
9.1 Summary of Works 169
9.1.1 Modelling the System 170
9.1.2 Schedule Generation 171
9.1.3 The Prototype Toolset 171
9.1.4 The Case Studies 172
9.1.5 Constraining the Schedule 173
9.2 Future Work 174
9.2.1 The Toolset 174
9.2.2 Practical Case Studies 174

Appendix
A Mathematical Symbols 176
B XML Description for Systems 180
B.1 Fluid Control System 180
B.2 Adaptive Cruise Control System 182
B.3 Robot Transport System 185
C User Guide of the Prototype Toolset 192
C.1 Loading XML Description into the Tool 192
C.2 Examining XML Description 194
C.3 Generating Timed Automata Models and a Schedule Graph 195
D Enhanced Olympic Boxing Scoring System Model 196

Bibliography 197

Abstract

The time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems. In such systems, the execution of distributed software components and the communication of messages between them take place in a fixed pattern and are scheduled in advance within a given scheduling round by a global scheduling policy. The principal obstacle in the design of time-triggered systems is the difficulty of finding the static schedule for all resources which satisfies constraints on the activities within the scheduling round, such as the meeting of deadlines. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide behaviour including messages transmitted on networks. Finding an efficient way of building an appropriate global schedule for a given system is a major research challenge.

This thesis proposes a novel approach to designing time-triggered schedules which is radically different from existing mathematical methods or algorithms for schedule generation. It entails the construction of timed automata to model the arrival and execution of software tasks and inter-task message communication for a system; the behaviour of an entire distributed system is thus a parallel composition of these timed automata models. A job comprises a sequence of tasks and messages; this expresses a system-wide transaction which may be distributed over a system of processors and networks. The job is formalized by a timed automata based on the principle that a task or message can be modelled by finite states and a clock variable. Temporal logic properties are formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages and adherence to deadlines. Schedules are computed by formally verifying that these properties hold for an evolution of the system; a successful schedule is simply a trace generated by the verifier, in this case the UPPAAL model-checking tool has been employed to perform

the behaviour verification. This approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists.

A prototype toolset has been developed to automate the proposed approach to create of timed automata models, undertake the analysis, extract schedules from traces and visualize the generated schedules. Two case studies, one of a cruise control system, the other a manufacturing cell system, are presented to demonstrate the applicability and usability of the approach and the application of the toolset. Finally, further constraints are considered in order to yield schedules with limited jitter, increased efficiency and system-wide properties.

Abstract

The time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems. In such systems, the execution of distributed software components and the communication of messages between them take place in a fixed pattern and are scheduled in advance within a given scheduling round by a global scheduling policy. The principal obstacle in the design of time-triggered systems is the difficulty of finding the static schedule for all resources which satisfies constraints on the activities within the scheduling round, such as the meeting of deadlines. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide behaviour including messages transmitted on networks. Finding an efficient way of building an appropriate global schedule for a given system is a major research challenge.

This thesis proposes a novel approach to designing time-triggered schedules which is radically different from existing mathematical methods or algorithms for schedule generation. It entails the construction of timed automata to model the arrival and execution of software tasks and inter-task message communication for a system; the behaviour of an entire distributed system is thus a parallel composition of these timed automata models. A job comprises a sequence of tasks and messages; this expresses a system-wide transaction which may be distributed over a system of processors and networks. The job is formalized by a timed automata based on the principle that a task or message can be modelled by finite states and a clock variable. Temporal logic properties are formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages and adherence to deadlines. Schedules are computed by formally verifying that these properties hold for an evolution of the system; a successful schedule is simply a trace generated by the verifier, in this case the UPPAAL model-checking tool has been employed to perform

the behaviour verification. This approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists.

A prototype toolset has been developed to automate the proposed approach to create of timed automata models, undertake the analysis, extract schedules from traces and visualize the generated schedules. Two case studies, one of a cruise control system, the other a manufacturing cell system, are presented to demonstrate the applicability and usability of the approach and the application of the toolset. Finally, further constraints are considered in order to yield schedules with limited jitter, increased efficiency and system-wide properties.

Abstract

The time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems. In such systems, the execution of distributed software components and the communication of messages between them take place in a fixed pattern and are scheduled in advance within a given scheduling round by a global scheduling policy. The principal obstacle in the design of time-triggered systems is the difficulty of finding the static schedule for all resources which satisfies constraints on the activities within the scheduling round, such as the meeting of deadlines. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide behaviour including messages transmitted on networks. Finding an efficient way of building an appropriate global schedule for a given system is a major research challenge.

This thesis proposes a novel approach to designing time-triggered schedules which is radically different from existing mathematical methods or algorithms for schedule generation. It entails the construction of timed automata to model the arrival and execution of software tasks and inter-task message communication for a system; the behaviour of an entire distributed system is thus a parallel composition of these timed automata models. A job comprises a sequence of tasks and messages; this expresses a system-wide transaction which may be distributed over a system of processors and networks. The job is formalized by a timed automata based on the principle that a task or message can be modelled by finite states and a clock variable. Temporal logic properties are formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages and adherence to deadlines. Schedules are computed by formally verifying that these properties hold for an evolution of the system; a successful schedule is simply a trace generated by the verifier, in this case the UPPAAL model-checking tool has been employed to perform

the behaviour verification. This approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists.

A prototype toolset has been developed to automate the proposed approach to create of timed automata models, undertake the analysis, extract schedules from traces and visualize the generated schedules. Two case studies, one of a cruise control system, the other a manufacturing cell system, are presented to demonstrate the applicability and usability of the approach and the application of the toolset. Finally, further constraints are considered in order to yield schedules with limited jitter, increased efficiency and system-wide properties.

Abstract

The time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems. In such systems, the execution of distributed software components and the communication of messages between them take place in a fixed pattern and are scheduled in advance within a given scheduling round by a global scheduling policy. The principal obstacle in the design of time-triggered systems is the difficulty of finding the static schedule for all resources which satisfies constraints on the activities within the scheduling round, such as the meeting of deadlines. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide behaviour including messages transmitted on networks. Finding an efficient way of building an appropriate global schedule for a given system is a major research challenge.

This thesis proposes a novel approach to designing time-triggered schedules which is radically different from existing mathematical methods or algorithms for schedule generation. It entails the construction of timed automata to model the arrival and execution of software tasks and inter-task message communication for a system; the behaviour of an entire distributed system is thus a parallel composition of these timed automata models. A job comprises a sequence of tasks and messages; this expresses a system-wide transaction which may be distributed over a system of processors and networks. The job is formalized by a timed automata based on the principle that a task or message can be modelled by finite states and a clock variable. Temporal logic properties are formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages and adherence to deadlines. Schedules are computed by formally verifying that these properties hold for an evolution of the system; a successful schedule is simply a trace generated by the verifier, in this case the UPPAAL model-checking tool has been employed to perform

the behaviour verification. This approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists.

A prototype toolset has been developed to automate the proposed approach to create of timed automata models, undertake the analysis, extract schedules from traces and visualize the generated schedules. Two case studies, one of a cruise control system, the other a manufacturing cell system, are presented to demonstrate the applicability and usability of the approach and the application of the toolset. Finally, further constraints are considered in order to yield schedules with limited jitter, increased efficiency and system-wide properties.

northumbria UNIVERSITY

GRADUATE SCHOOL

THESIS/SUBMISSION DECLARATION

Title of thesis/submission: Automatic Schedule Computation for Distributed Real-Time Systems using Timed Automata

Author: YOUNG SAENG PARK

Research undertaken in collaboration with: the School of Computing, Engineering & Information Sciences

I, YOUNG SAENG PARK, **the undersigned, claim copyright of the above described thesis/submission and declare that no quotation from it or information from it may be published without my prior written consent.**

Signed: .....

Date: 31/01/2008

BRITISH LIBRARY

Document Supply

Submission of Theses to the British Library

The British Library is asking the universities' co-operation in requiring doctoral degree candidates to carry out the following:

Complete a British Library agreement form as part of their standard procedure. The form;

i asks authors to provide personal data and thesis data as it will appear in catalogue records and alerting media.

ii asks authors to sign authorisation enabling the British Library to produce copies for loan or sale.

iii provides for payment of royalties by the British Library to the author. The royalty will be 10% of revenue received from sales of a second or subsequent copy during any period of one year. The royalty will be paid annually in April, and it is the author's responsibility to keep the British Library informed of any change in address.

iv sets out minimum standards of physical presentation. The text supplied must be the examined printed text; options for acceptance of electronic full-text are under review.

Supply the British Library with a photocopy of the title page and abstract with the completed agreement form.

British Theses and Copyright

Author's Rights

Is the British Library proposing to publish British Theses? No.

A British thesis is an unpublished work within the meaning of the *Copyright, Designs and Patents Act*, 1988, *(s.175)*. Copies of a thesis cannot be issued to the public without the copyright owner's consent *(s.16)*

British theses are protected under the Act as unpublished works. The Act prohibits re-publication of any significant part of a thesis by a third party without the copyright owner's consent. The British Library produces single copies for loan or retention in response to specific demand. Issuing single copies on demand does not constitute publishing.

Do the British Library's proposals compromise the author's rights? No.

The copyright of a thesis belongs to the author but this ownership may be assigned by written agreement either specifically or as part of an undertaking between the researcher and the awarding institution when the course of research was entered upon. If the awarding institution actually employs the researcher to undertake the work, the copyright belongs automatically to the awarding institution *(s.11)*

Under the British Library's scheme the author is asked to sign an agreement permitting the British Library to copy his/her thesis on demand. The copies will either be lent or sold to individuals or libraries.

Copyright ownership is unaffected. The copy supplied has the same protection under the Act as the original copy.

The British Library undertakes that in *every* copy supplied, either for loan or retention, the following statement will be included:

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Document Supply

Boston Spa, Wetherby,
West Yorkshire
LS23 7BQ

BRITISH LIBRARY DOCTORAL THESIS AGREEMENT FORM

The British Thesis Service is designed to promote awareness of and improve access to the results of publicly funded British Doctoral Research.

Records of theses in the scheme are available for searching in The British Library Public Catalogue (www.blpc.bl.uk) and the *Index to Theses* published by Expert Information Ltd (www.theses.com).

On demand access is provided for individual researchers and libraries from a single, central collection of more than 170,000 doctoral theses.
See www.bl.uk/britishthesis for more information

Access Agreement

Through my *university/college/department, I agree to supply the British Library Document Supply Centre, with a copy of my thesis.

I agree that my thesis may be copied on demand for loan or sale by the British Library, or its agents, to requesting libraries (who may add the copy to their collection for loan or consultation) or individuals. I understand that any copies of my thesis will contain the following statement:

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

I confirm that the thesis and abstracts are my original work, that further copying will not infringe any rights of others, and that I have the right to make these authorisations. Other publication rights may be granted as I choose.

The British Library agrees to pay me a royalty of ten percent on any sales of the second and subsequent copies of my thesis per year. The royalty will be paid annually in April.

In order to be eligible for such royalties, I agree that my obligation is to notify the British Library of any change of address.

* please delete as applicable.

INSTRUCTIONS FOR BRITISH DOCTORAL THESIS FORM

Please complete this agreement carefully, so that your thesis can be made available as rapidly as possible. Please type or print in black, and look through the instructions which follow for explanations concerning the Agreement Form.

ITEM 1:

Full name. This should be as shown on your title page, and as registered with the awarding body.

ITEM 2:

Future mailing address. This is the address at which you can be reached after you have completed your degree requirements. It will be used in mailing any royalties due from the sale of your thesis.

ITEM 3a:

Please enter your university name, plus college or department. For example: University of York, Department of Computer Science.

ITEM 4:

Enter the name(s) of any sponsoring body other than those in 3a.

ITEM 8:

Additional keywords. If your thesis requires additional keywords for identification which are not included in the title, please add up to five keywords.

ITEM 9:

Select from the Subject Categories section in this leaflet the one category that best relates to the subject content of your thesis. Enter the corresponding code in the boxes and the category title next to it. If you wish, you may indicate up to two additional subject codes.

If you have any questions please contact:

The British Thesis Service
The British Library
Document Supply Centre

Tel: 01937 546229
Fax: 01937 546286
E-mail: dsc-british-thesis-service@bl.uk

STANDARDS

The physical presentation of your thesis should be in accordance with the following specifications

- The copy must be legible. The size of character used in the main text, including displayed matter and notes, should be not less than 2.0mm for capitals and 1.5mm for x-height (height of lower case x)
- Paper should be size A4, white and within the range 70g/m^2 to 100g/m^2.
- Text should be single sided - right hand pages (rectos) only.
- The margin at the binding edge of the page should not be less than 40mm. Other margins should not be less than 15mm. Running heads and page numbers should be within the recommended margins.

The recommendations as set out in the withdrawn BS 4821 : 1990 remain good practice.

IMPORTANT:

If your thesis is not produced according to these standards, it may not be possible to include it in the scheme.

BRITISH DOCTORAL THESIS AGREEMENT FORM

Please type or print in black ink

Personal Data

1 Surname PARK

Forenames YOUNG SAENG

2 Present mailing address

young-saeng.park@unn.ac.uk

Future mailing address

com1984@hanmail.net

Effective date for future address

Doctoral Degree Data

3 Full name of University conferring degree, and college or division if appropriate

NORTHUMBRIA UNIVERSITY, SCHOOL OF COMPUTING, ENGINEERING & INFORMATION SCIENCES

4 Name of co-sponsoring body(ies) (if any)

5 Abbreviations for degree awarded

PhD

6 Date degree awarded 22/01/2008

7 Title of thesis Automatic Schedule Computation for Distributed Real-Time Systems using Timed Automata

8 List up to five additional descriptive keywords or short phrases not in your thesis title to help subject access.

a TIMED AUTOMATA

b SCHEDULE COMPUTATION

c DISTRIBUTED REAL-[redacted]TIME SYSTEM

d REAL-TIME SCHEDULE

e

9 Subject category for your thesis. Enter a code from the Subject Category list overleaf and write in the category selected. You may enter two additional categories and/or codes on the extra lines provided.

0 9 H	COMPUTER SOFTWARE; PROGRAMMING
0 9 J	CONTROL SYSTEMS; CONTROL THEORY

10 Language of text (if not English)

IMPORTANT:
You must sign the Access Agreement on page 1 of this leaflet and enclose it with this form and your thesis

Subject Categories

The British Library will include details of your thesis in printed and electronic awareness media. Entries are arranged by broad, general subject categories. Please select the one subject category which most nearly describes the contents of your thesis. Enter the corresponding code in the spaces provided in Item 9 of the Agreement Form.

Aeronautics
- 010 Aeronautics, general
- 01A Aerodynamics
- 01B Aircraft operations; aircraft safety; aircraft accidents; air traffic control
- 01C Aircraft; aircraft components
- 01D Aircraft flight control; aircraft instrumentation
- 01E Airport operations
- 01F Parachutes; decelerators

Agriculture, plant and veterinary sciences
- 020 Agriculture, plant and veterinary sciences, general
- 02A Agricultural chemistry; fertilisers; animal feed; silage; pesticides; soil conditioners
- 02B Agricultural economics; agricultural markets
- 02C Agricultural engineering
- 02D Agronomy; crop production; crop diseases; horticulture
- 02E Animal husbandry; farm animals; pets
- 02F Forestry; agroforestry; sustainable forestry
- 02G Veterinary sciences; veterinary medicine
- 02H Aquaculture; fisheries; fishing

Environmental pollution, protection and control
- 030 Environmental pollution, protection and control, general
- 03A Air pollution; emissions; acid rain
- 03B Noise pollution
- 03C Water pollution; oil pollution; sewage treatment; water treatment
- 03D Land pollution; soil pollution
- 03E Radioactive pollution; nuclear waste
- 03F Solid waste pollution; waste disposal; landfills
- 03G Environmental health; environmental safety
- 03H Environmental law; environmental regulations
- 03I Waste recycling (non-nuclear); waste recovery
- 03J Nuclear waste reprocessing

Humanities, psychology and social sciences
- 050 Humanities, psychology and social sciences, general
- 05A Management; administration; business studies
- 05B Information science; librarianship
- 05C Ergonomics
- 05D Economics; economic theory
- 05E History
- 05F Archaeology
- 05G Anthropology; folklore; ethnology
- 05H Philosophy; theology; religion
- 05I Law; law enforcement; penal administration
- 05J Political science; public administration
- 05K Linguistics
- 05L Literature; mass media; performing arts
- 05M Sport; Recreation; tourism
- 05N Arts; crafts
- 05O Architecture
- 05P Education; training
- 05Q Psychology
- 05R Sociology; social studies; welfare studies; social services
- 05S Labour studies
- 05T Health services; health administration; community care services
- 05U Housing provision; property
- 05V Urban planning; rural planning; transport planning; countryside conservation
- 05W Demography; population studies
- 05X Internal and EU commerce; domestic marketing; consumer affairs
- 05Y International commerce; international marketing; international trade
- 05Z Banking; finance; taxation

Biological and medical sciences
- 060 Biological and medical sciences, general
- 06A Biochemistry
- 06B Bioengineering; biomedical engineering; biotechnology; biochemical engineering
- 06C Human anatomy; human histology
- 06D Bionics
- 06E Medicine
- 06F Ecology
- 06G Escape; rescue; survival
- 06H Food technology; food microbiology
- 06I Hygiene; sanitation
- 06J Industrial medicine; occupational health
- 06K Life-support systems
- 06L Medical equipment; hospital equipment; medical diagnostic equipment
- 06M Microbiology
- 06N Personnel selection; employee fitness
- 06O Pharmacology; pharmacy; pharmaceutical chemistry
- 06P Physiology
- 06Q Protective equipment; protective clothing
- 06R Radiobiology; radiation biology
- 06S Stress physiology, human; aerospace medicine
- 06T Toxicology; poisons
- 06U Wounds; Injuries; trauma medicine
- 06V Genetics; cytology; molecular biology
- 06W Botany
- 06X Zoology
- 06Y Biophysics
- 06Z Alternative medicine

Chemistry
- 070 Chemistry, general
- 07A Chemical engineering; industrial chemistry
- 07B Inorganic chemistry
- 07C Organic chemistry
- 07D Physical chemistry
- 07E Nuclear chemistry; radiochemistry
- 07F Analytical chemistry

Earth and atmospheric sciences
- 080 Earth and atmospheric sciences, general
- 08A Oceanography
- 08B Hydrology; limnology
- 08C Glaciology; snow; ice; permafrost
- 08D Geography
- 08E Geology; mineralogy; sedimentology
- 08F Seismology; earthquakes
- 08G Volcanology; plate tectonics
- 08H Earth, interior structure
- 08I Geochemistry
- 08J Geomagnetism; geodesy; cartography
- 08K Soil Science; pedology
- 08L Mining
- 08M Atmospheric sciences
- 08N Meteorology; climatology
- 08Q Biosphere

Electronics and electrical engineering, computer science
- 090 Electronics and electrical engineering, computer science
- 09A Components
- 09B Circuits
- 09C Electronic devices; electromechanical devices
- 09D Optoelectronics
- 09E Power transmission; signal transmission
- 09F Electrometry; electronic test equipment
- 09G Computer hardware
- 09H Computer software; programming
- 09I Control systems; control theory
- 09J Information theory; coding theory; signal processing
- 09K Pattern recognition; image processing
- 09L Computer-Aided Design (CAD)
- 09M Computer Aided Manufacturing (CAM)
- 09N Robotics
- 09O Applications of computer science; business data processing
- 09P Artificial Intelligence

Energy and power
- 100 Energy and power, general
- 10A Coal
- 10B Oil
- 10C Natural gas
- 10D Oil shales; coal shales; tar sands
- 10E Fission fuels
- 10F Fusion fuels
- 10G Isotope technology; radiation source technology
- 10H Hydrogen
- 10I Other synthetic and natural fuels
- 10J Hydro energy
- 10K Solar energy
- 10L Geothermal energy
- 10M Wave power; tidal power
- 10N Wind power
- 10O Nuclear power plants
- 10P Nuclear reactor technology
- 10Q Energy storage
- 10R Direct energy conversion; Fuel cells
- 10S Energy conservation; Energy consumption
- 10T Special purpose power plants
- 10U Biomass energy
- 10V Thermodynamic cycles
- 10W Conventional power plants

Materials
- 110 Materials, general
- 11A Adhesives; sealants
- 11B Ceramics; refractories; Glasses
- 11C Coatings; paints; finishes
- 11D Composites
- 11E Fibres; textiles
- 11F Metallurgy; metallography
- 11G Miscellaneous materials
- 11H Oils; Lubricants; hydraulic fluids
- 11I Plastics
- 11J Elastomers
- 11K Solvents; cleaners; abrasives
- 11L Wood; paper
- 11M Material degradation; corrosion; fracture mechanics

Mathematical sciences
- 120 Mathematical sciences, general
- 12A Pure mathematics
- 12B Statistics; operations research
- 12C Applied mathematics

Mechanical, industrial, civil and marine engineering
- 130 Mechanical, industrial, civil and marine engineering; general
- 13A Air conditioning; heating; lighting; ventilation; refrigeration
- 13B Civil engineering
- 13C Construction equipment; building materials
- 13D Containers; Packaging
- 13E Couplings; fittings; fasteners; joints
- 13F Ground transport systems
- 13G Hydraulic systems; pneumatic systems
- 13H Industrial processes; manufacturing processes
- 13I Machinery; tools
- 13J Marine engineering; offshore engineering
- 13K Pumps; filters; pipes; tubing; valves; pressure vessels
- 13L Safety engineering
- 13M Structural engineering
- 13N Building technology

Methods and equipment
- 140 Methods and equipment, general
- 14A Cost effectiveness; cost-benefit analysis
- 14B Laboratories; test facilities; test equipment
- 14C Recording equipment
- 14D Equipment reliability; quality control
- 14E Reprographics; photographic processes

Military sciences
- 150 Military sciences, general
- 15A Antisubmarine warfare
- 15B Chemical warfare; biological warfare
- 15C Defence
- 15D Military intelligence
- 15E Military logistics
- 15F Nuclear warfare
- 15G Military operations; military strategy; military tactics

Missile technology
- 160 Missile technology, general
- 16A Missile launching; missile ground support
- 16B Missile trajectories
- 16C Missile warheads; missile fuses
- 16D Missiles

Navigation, communications, detection and countermeasures
- 170 Navigation, communications, detection and countermeasures; general
- 17A Acoustic detection; sonar
- 17B Communication systems; telecommunications
- 17C Direction finding
- 17D Electronic countermeasures; acoustic countermeasures
- 17E Infrared detection; ultraviolet detection
- 17F Magnetic detection
- 17G Navigation
- 17H Optical detection
- 17I Radar detection
- 17J Seismic detection

Ordnance
- 190 Ordnance, general
- 19A Pyrotechnics; explosives; ammunition
- 19B Bombs
- 19C Combat vehicles
- 19D Explosions; ballistics; armour
- 19E Fire control; bombing systems
- 19F Guns
- 19G Rockets
- 19H Underwater ordnance

Physics
- 200 Physics, general
- 20A Theoretical physics
- 20B Elementary particles; high energy physics
- 20C Nuclear physics; particle accelerators
- 20D Atomic physics; molecular physics
- 20E Optics; masers; lasers
- 20F Acoustics; vibrations; noise analysis
- 20G Thermodynamics
- 20H Metrology
- 20I Fluid mechanics
- 20J Plasma physics; gas discharges
- 20K Solid-state physics
- 20N Astronomy; celestial mechanics
- 20O Astrophysics
- 20P Cosmic rays
- 20Q Astrogeology; planetary research

Propulsion and fuels
- 210 Propulsion and fuels, general
- 21A Air-breathing engines
- 21B Combustion; ignition
- 21C Electric propulsion
- 21D Fuels
- 21E Jet turbine engines; gas turbine engines
- 21F Nuclear propulsion
- 21G Reciprocating engines
- 21H Rocket motors
- 21I Rocket propellants

Space technology
- 20 Space technology, general
- 22A Astronautics
- 22B Manned spacecraft
- 22C Spacecraft trajectories; spacecraft re-entry
- 22D Spacecraft launching; ground support
- 22E Unmanned spacecraft; satellites
- 22F Extraterrestrial exploration

Automatic Schedule Computation for Distributed Real-Time Systems using Timed Automata

Young Saeng Park

Doctor of Philosophy

2007

Automatic Schedule Computation for Distributed Real-Time Systems using Timed Automata

Young Saeng Park

A thesis submitted in partial fulfilment
of the requirements of the
University of Northumbria at Newcastle
for the degree of
Doctor of Philosophy

Research undertaken in the School of Computing, Engineering &
Information Sciences

March 2007

Abstract

The time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems. In such systems, the execution of distributed software components and the communication of messages between them take place in a fixed pattern and are scheduled in advance within a given scheduling round by a global scheduling policy. The principal obstacle in the design of time-triggered systems is the difficulty of finding the static schedule for all resources which satisfies constraints on the activities within the scheduling round, such as the meeting of deadlines. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide behaviour including messages transmitted on networks. Finding an efficient way of building an appropriate global schedule for a given system is a major research challenge.

This thesis proposes a novel approach to designing time-triggered schedules which is radically different from existing mathematical methods or algorithms for schedule generation. It entails the construction of timed automata to model the arrival and execution of software tasks and inter-task message communication for a system; the behaviour of an entire distributed system is thus a parallel composition of these timed automata models. A job comprises a sequence of tasks and messages; this expresses a system-wide transaction which may be distributed over a system of processors and networks. The job is formalized by a timed automata based on the principle that a task or message can be modelled by finite states and a clock variable. Temporal logic properties are formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages and adherence to deadlines. Schedules are computed by formally verifying that these properties hold for an evolution of the system; a successful schedule is simply a trace generated by the verifier, in this case the UPPAAL model-checking tool has been employed to perform

the behaviour verification. This approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists.

A prototype toolset has been developed to automate the proposed approach to create of timed automata models, undertake the analysis, extract schedules from traces and visualize the generated schedules. Two case studies, one of a cruise control system, the other a manufacturing cell system, are presented to demonstrate the applicability and usability of the approach and the application of the toolset. Finally, further constraints are considered in order to yield schedules with limited jitter, increased efficiency and system-wide properties.

Contents

1 Introduction

1.1 Background 1
1.1.1 Real-Time Systems 1
1.1.2 Scheduling Difficulty in Distributed Real-Time Systems 3
1.1.3 Timed Automata 5
1.2 The Central Proposition 6
1.3 Summary of Contributions 8
1.4 Organisation of the Thesis 8

2 Background of Real-Time Scheduling Theory

2.1 Real-Time Scheduling Principle 10
2.1.1 Types of Tasks 11
2.1.2 Classification of Scheduling 12
2.1.2.1 Static Cyclic Scheduling 12
2.1.2.2 Fixed Priority Scheduling 14
2.1.2.3 Dynamic Priority Scheduling 16
2.1.2.4 Comparison between Fixed Priority and Static Cyclic Scheduling 17
2.1.3 Schedulability Analysis 18
2.1.3.1 Processor Utilization Analysis 18
2.1.3.2 Response Time Analysis 21
2.1.3.3 Schedulability Analysis in Distributed Real-Time Systems 23
2.2 Event-Triggered and Time-Triggered Architecture 24
2.2.1 Predictability 25
2.2.2 Resource Utilization 25
2.2.3 Extensibility 26
2.2.4 Testability 27

2.3 Global Scheduling 27
2.3.1 Heuristic Approach 28
2.3.1.1 List Scheduling 28
2.3.1.2 Iterative Deepening A* 29
2.3.2 Simulated Annealing 31
2.3.3 Genetic Algorithm 32
2.3.4 Tabu Search 34
2.4 Summary 35

3 Background of Timed Automata
3.1 Timed Automata 37
3.1.1 Clocks 38
3.1.2 Clock Constraints 39
3.1.3 Timed Automaton 39
3.1.4 Composition of Timed Automata 40
3.1.5 State-Transition System 40
3.1.6 An Example of a Timed Automata Model 41
3.2 Techniques for the Verification of Timed Automata 42
3.2.1 Clock Regions 42
3.2.2 Clock Zones 44
3.3 The Application of Timed Automata to Schedulability 45
3.4 Summary 49

4 Scheduling with Timed Automata
4.1 Formal Definitions of the Terms 50
4.1.1 System 51
4.1.2 Task and Message 51
4.1.3 Precedence Constraint 52
4.1.4 Job 53
4.1.5 An Example of Formal Expression 53
4.2 Design Principle of Schedules with Timed Automata 54
4.2.1 A Timed Automaton for a Task and Message 55
4.2.2 A Timed Automaton for a Job 55
4.2.3 A Timed Automaton for a System Schedule 57
4.2.4 An Example of a Timed Automata model for a Job 58

4.3 Feasible Schedules from Timed Automata 61
4.3.1 Feasible Schedules 61
4.3.2 Finding Feasible Schedules 62
4.4 Summary 64

5 Efficient Timed Automata Models for Scheduling
5.1 Overview of UPPAAL 65
5.1.1 UPPAAL 65
5.1.2 Syntax 66
5.1.3 An Example of a UPPAAL model: Olympic Boxing Scoring System 67
5.1.3.1 Description 67
5.1.3.2 Modelling in UPPAAL 68
5.1.4 Verification 71
5.1.5 Model-Checking and State Explosion Problem 72
5.2 Scheduling Models with UPPAAL 73
5.2.1 Model One 73
5.2.2 Model Two 75
5.2.3 Model Three 76
5.3 Evaluation of the Models 78
5.4 The Limitation of the Timed Automata Model 82
5.5 Summary 84

6 Scheduling Generation with UPPAAL Models
6.1 The Proposed Approach 85
6.2 Prototype Toolset Based on the Approach 89
6.2.1 Fluid Control System 89
6.2.2 Preprocessor 91
6.2.2.1 Description of a System with XML 91
6.2.2.2 Description of a Processor with XML 92
6.2.2.3 Description of a Network with XML 93
6.2.2.4 Description of a Job with XML 95
6.2.2.5 Description of the FC System with XML in the Tool 96
6.2.3 Analyser Engine 96
6.2.3.1 Timed Automata Generator 97
6.2.3.2 Trace Analyser 101

6.2.3.3 Graphic Viewer 102
6.2.3.4 Schedule Extractor 104
6.3 Comparison of the Proposed Approach 105
6.4 Summary 108

7 Case Studies

7.1 Case Study: Adaptive Cruise Control System 110
7.1.1 Description 110
7.1.2 Structure 111
7.1.3 Operation 112
7.1.4 System Description with XML 115
7.1.5 Outputs from the Prototype Toolset 117
7.1.6 Schedule for the ACC System 120
7.2 Case Study: Robot Transport System 121
7.2.1 Description 122
7.2.2 Structure 123
7.2.3 Operation 125
7.2.4 Description with XML 130
7.2.5 Experimental Studies 132
7.2.5.1 Influence of Arbitrary Time Units – Study A 132
7.2.5.2 Influence of Appearance of Jobs – Study B 134
7.2.5.3 Influence of Utilisation of Processors and Networks – Study C 137
7.2.5.4 Influence of Different Timing Values – Study D 140
7.3 Summary 142

8 Further Schedule Constraints

8.1 Schedules Considering Jitter 143
8.1.1 Jitter Control with Temporal Properties – Method A 146
8.1.2 Jitter Control with UPPAAL Models – Method B 149
8.1.3 Comparison between using Temporal Properties and UPPAAL Models 151
8.2 Scheduling Compaction 152
8.3 Precedence Constraints between Jobs 159
8.3.1 Precedence Constraint with Any Time – Case ① 161
8.3.2 Precedence Constraint without Time Delay – Case ② 162

8.3.3 Precedence Constraint with Offset Time – Case ③ 162
8.3.4 Experiments with Precedence Constraints 163
8.4 Summary 167

9 Conclusion and Future Work
9.1 Summary of Works 169
9.1.1 Modelling the System 170
9.1.2 Schedule Generation 171
9.1.3 The Prototype Toolset 171
9.1.4 The Case Studies 172
9.1.5 Constraining the Schedule 173
9.2 Future Work 174
9.2.1 The Toolset 174
9.2.2 Practical Case Studies 174

Appendix
A Mathematical Symbols 176
B XML Description for Systems 180
B.1 Fluid Control System 180
B.2 Adaptive Cruise Control System 182
B.3 Robot Transport System 185
C User Guide of the Prototype Toolset 192
C.1 Loading XML Description into the Tool 192
C.2 Examining XML Description 194
C.3 Generating Timed Automata Models and a Schedule Graph 195
D Enhanced Olympic Boxing Scoring System Model 196

Bibliography 197

Acknowledgements

First and foremost I would like to thank my supervisors, Adrian Robson and William Henderson for guiding me with great enthusiasm and technical knowledge. During my years as a PhD student, they have not only inspired me to investigate various interesting topics, but also pointed me to good directions with reasonable comments. In particular, the review of this thesis from them was really invaluable during the period of writing the thesis. I would not forget having a coffee and a lot of talks in the supervision every Friday. I also want to thank the University of Northumbria which provided the major part of my research funding.

I would especially like to thank my wife, Eun-Mi Son, who has overcome many arduous difficulties with me. I acknowledge that she tasted every piece of sweetness and bitterness more than me. For many days and nights, she sacrificed her time to give me comfort and let me concentrate on my study. Without her love, care and support, finishing the study would never have been possible. Also, I like to share this delightful moment with my little baby, Jae-Hyun Park, who has always given me a big smile every day. His smile was and is still a source of my living and encourages me to enthusiastically work hard.

I would like to have an opportunity to express my deepest gratitude to the people who have expected me to successfully finish my study and take a degree: my mother, my father-in-law, my mother-in-law and the rest of my family for always supporting and believing in me. Lastly, I would like to sincerely thank my father, the very special person who was expecting the success than others. I believe that he would be delighted and enjoy this moment in heaven. However, I would never be forgiven that I did not have much time with him before he died, by the excuses of studying and living far away. This thesis is dedicated to my father. Thanks.

Declaration

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work.

Name:

Signature:

Date:

Chapter 1

Introduction

This thesis focuses on aspects related to designing schedules in time-triggered architectures, in particular, for distributed real-time systems. The principal aim of this research is to find a more rigorous approach to automatically generate schedules for such systems. An outcome of this thesis is the introduction of a novel approach using timed automata models to compute the schedules for systems and furthermore to generate together not only the local schedules but also the global schedules. In this chapter, some of the basic concepts used throughout this thesis are briefly presented, followed by a summary of the contributions and an overview of the structure of this thesis.

1.1 Background

1.1.1 Real-Time Systems

Real-time systems are becoming crucial in a wide variety of fields in providing effective functional capabilities for a variety of purposes. This is because the computers (or control devices) are getting not only faster and cheaper but also smaller and lighter. Examples of real-time systems are found everywhere, including washing machines, mobile-phones in our everyday life, and also air control systems, nuclear power plant control systems, railway switching systems and automotive electronics. In general, real-time systems are defined as those in which the correctness of the systems depends not only on the logical results but also on the time at which the results are produced [AB90, BW01, But97, Kop97, WT95]. Missing a required time (or deadline) may lead to

system failure e.g. wrong information, serious functional problem, financial damage or catastrophic loss of human life in more serious cases.

Depending on the consequences of the failures, real-time systems are often categorised as either *soft* or *hard*. Systems are said to be soft if missing a deadline does not cause serious damage and the systems will still work providing functionally correct behaviour, whereas systems are said to be hard when missing a deadline may result in catastrophic consequences [GR04]. For example, a real-time audio system is known as a soft real-time because the delay of data stream may still be fine but just cause a loss in the quality of the system. Nuclear power plant control is a hard real-time system because timeliness is absolutely essential rather than losing the quality. Especially in hard real-time systems, it is more important to guarantee the correctness of systems by promising all tasks meet their deadline on time even whether under maximum load or not, and thus hard real-time systems are often safety-critical systems which require highly reliable, available service and highly stringent timing constraints.

More and more real-time systems are implemented on multiprocessors; many might have to be physically distributed due to imposed constraints. In many fields requiring high performance and large scale, *distributed real-time systems* are introduced as good solutions [Stn92]. The typical architecture comprises a number of processors, one or more interconnection networks and various tasks belonging to each processor [Kop97]. Each task in the systems may independently work with its own purpose but also cooperate with other tasks, perhaps located in other processors by message passing. These tasks working with other tasks collectively are called here as a *job* (in spite of having many different terms employed in the literature). A typical example of a job is found in a monitoring system which may involve three tasks: sampling sensor data; sending the sampled data over the network; displaying the data. In a job, it is important to guarantee not only that each task has to be completed before its deadline but also that each message has to be reached to its associated task before the task starts. Thus, when considering schedules for distributed real-time systems [Mar00], it is important to determine timely scheduling of tasks within each processor (local scheduling), and scheduling of tasks over all processors and messages on networks (global scheduling).

There are currently two fundamentally different designing architectures for real-time systems: one is based on *event-triggered* and the other is based on *time-triggered*

architecture. All activities in an event-triggered architecture, such as starting tasks or sending a message, are driven by the occurrences of events and not by the passage of time, while a time-triggered architecture controls all activities by a recurring clock tick and at pre-determined points in time. The major advantage of event-triggered architectures is that they enable immediate response to the occurring events coming from the outside environment or inside systems and thus they have higher flexibility by adapting any demand quickly without considering the complete system. On the contrary, the time-triggered architectures have a drawback which is the lack on the flexibility, as required all activities must be known in advance and then behave strictly under a fixed pattern. However, the drawback allows more determinism to the architectures and consequently the determinism gives better safety and reliability to the architectures [AG04].

Depending on the requirement of systems such as flexibility, scalability, safety, reliability, etc, it is critical to select an adequate system, satisfying the requirements among various types of real-time systems. In particular, with regard to the increase of large scale systems for high performance, higher reliability and stringent timing constraints over the last decade such as automotive electronics, aerospace industry, etc, the real-time systems considering in this thesis are distributed real-time systems using a time-triggered architecture.

1.1.2 Scheduling Difficulty in Distributed Real-Time Systems

In real-time systems using a time-triggered architecture, all activities in the systems which act strictly under fixed pattern are scheduled in advance by the available knowledge based on a static design [RFA93, SW97, SW98]. This adds determinism and more reliability to the systems. However, there are problems in providing the required knowledge, in particular, to build the knowledge of schedules for distributed real-time systems in the time-triggered architecture. It is a major challenge for engineers to find an efficient way to build the appropriate schedules and generate them automatically.

The main obstacle in building appropriate schedules is how to find schedules within the given specific time window and to satisfy other constraints such as execution time, deadline, etc. Besides, there are further demands when considering activities consisting of many distributed tasks with precedence constraints, known a *job* here. In this case, it

is required to consider not only a schedule within each processor, but also a global schedule of overall processors including messages transmitted on networks [Tha02]. For example, consider a fluid control system as an example of distributed real-time systems using a time-triggered architecture. The fluid system consists of the two processors separately located on different computer nodes performing flow measurement and valve control; these are connected by a real-time communication network. The flow control node is connected to the flow sensor in order to read the rate of fluid flow; the valve node controls the valve via an actuator according to a command message transmitted by the flow node (see Figure 1.1).

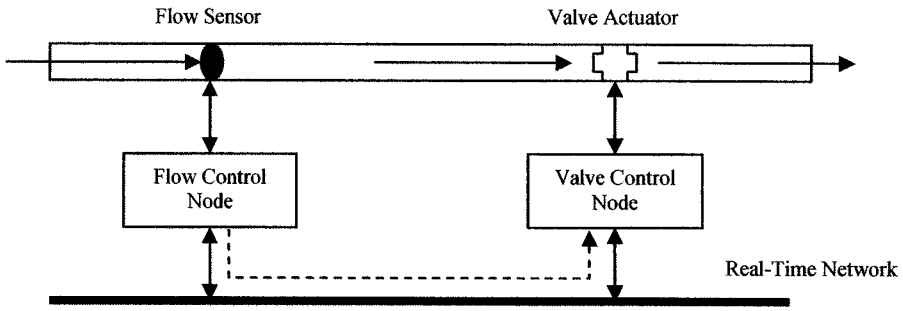


Figure 1.1 Fluid Control System

Suppose that the system has two jobs. One of the jobs has to transmit the current flow rate from the flow node to the valve node every 100 time-units; two tasks with exactly the same frequency located on different nodes are involved in this job; they communicate by message passing on the network. The other job has a similar work transmitting alarm data but this time the job occurs every 50 time-units. In this example, it is quite difficult to find a satisfied schedule for each node with considering these different job patterns on the system. A successful schedule on the flow node might cause a failure on the valve node. Consequently, the schedule for the flow node and the value node, and the schedule for the messages on the network should be considered at the same time. In particular, it is required to produce the overall scheduling within a specific response time window for the fluid system. In cases of considering more jobs in the system, the scheduling problem gets more and more complicated. Finding schedules for many jobs in distributed real-time systems is, in general, known as NP-hard problems [Bur91, But97].

There have been many studies undertaken on single processor architectures in order to predict and meet timing requirements. Scheduling of distributed real-time systems in a time-triggered architecture has received less attention and is less complete. Therefore, it is desirable to find a cost-effective solution for distributed real-time systems, in particular, using a time-triggered architecture. The solution should produce not only local schedules but also global schedules for both processors and networks at the same time [AS99, BL92, MSM+96, Nae98, PEP02].

1.1.3 Timed Automata

Within the last decade, *model checking* has developed as useful technique for verifying finite-state systems. The basic idea of model checking is to verify the correctness of systems via efficient algorithms executed using computer tools [BBF+01, Yov96]. Model checking for systems typically employs an exhaustive state space search; it explores the whole reachable state – space of a system – examining all the possible behaviours of the system. One of these techniques is known as *reachability analysis*. In model checking, it is important to make well-described models (the possible system behaviour) and well-described requirements for systems (the desirable system behaviour) as input values to enable successful verification [BHV00, Kat99, Pet99].

In the last few years, model checking has been extended and applied to examine the temporal behaviour of real-time systems since *timed automata* are introduced as means to model real-time systems with finite-states and a set of clocks. Traditional model checking does not admit an explicit modelling of time and is, therefore, not suitable for the analysis of real-time systems in which not only the logical result of the correct computation is modelled but also the respect of strict timing constraints such as execution times, period, response time, delay and so on. Accordingly, timed automata were firstly introduced by Alur *et al.* [AD90, AD94] as an extension of the automata-theoretic approach with time. Timed automata are equipped with a finite set of real-valued clock variables to measure the elapse of time [Kat99]. All clocks proceed at the same rate and the value of a clock denotes how much time elapsed since it was initialised. Timed automata also have a set of *edges* (or locations) and *transitions*. Changing between edges is possible via transitions and controlled using *guards* and *invariants* which are a condition for transitions and edges respectively. Since introduced,

timed automata have been intensively applied to solve real-life research problems in computer science, particularly in time-critical systems ranging from communication protocols to safety-critical systems. As a consequence, numerous model-checking tools have been introduced such as *KRONOS* [BDM+98] and *UPPAAL* [DBL+03, Hen02, LPY97, LPY98].

In particular, UPPAAL is a well-known timed automata tool for symbolic model checking of real-time systems; it has been developed jointly by Uppsala University and Aalborg University. It supports simulation and formal verification of the behaviour of systems by checking invariant and reachability properties, and even provides facilities to detect deadlocks. It is suitable for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communication through channels or shared variables [LPY97]. Although UPPAAL implements a similar notion of timed automata developed by Alur *et al.* [AD90], the main goal of it is to provide a more expressive and efficient way to modelling systems. Thus, they extend timed automata with more general data types such as integer, boolean and clock variables.

UPPAAL has been applied to several industrial case studies such as real-time protocols, multi-media synchronization protocols, real-time controller and proving the correctness of plants [Feh99] and so on. Havelund *et al.* [HSL+97] apply it in the context of audio and video protocols and used it to model and prove the correctness of control protocols. UPPAAL has been used to formally verify a time division multiple access (TDMA) based protocol intended for local area networks that operate in modern vehicles by Lönn *et al.* [LA99]. It has also been used to help supporting the development of a system in automotive industry.

1.2 The Central Proposition

This research concerns schedule generation for distributed real-time systems in a time-triggered architecture. In such systems, the execution of distributed software tasks and the communication of messages between them takes place in a fixed pattern that is scheduled in advance. The schedule for the system has to consider not only the requirements on each processor but also the global requirements of system-wide

behaviour. This makes it difficult to find a static schedule for all resources satisfying constraints on the activities within the scheduling round.

There are a number of different algorithms and methods in the literature for solving the scheduling problem. Many of these are based on an approximation, such as a designer's intuition or experience, rather than absolute accuracy. Consequently, these approaches may find a feasible schedule which is good enough or even close to optimal among all possible schedules, but they do not guarantee the quality of schedules. Indeed, they may not find a schedule even though one exists for the system.

> **Hypothesis:** *Model checking is capable of yielding schedules for difficult scheduling problems in time-triggered architectures. It enables the identification of schedules and guarantees the quality.*

In this research, model checking is used to explore all reachable states in a system; allowing all scheduling behaviours to be examined. This provides an exceptional capability to find a feasible schedule, compared with the existing algorithms and methods. It guarantees to find a feasible schedule if one exists and will fail to find any schedule if none exists. However, there remains an issue of the state explosion problem particularly when considering complex and large systems.

Here, a novel approach is proposed to generate schedules for distributed real-time systems in a time-triggered architecture. Briefly, the approach constructs timed automata models and temporal logic properties representing the behaviours and requirements of the systems. It entails the construction of a model of the arrival and execution of software tasks and inter-task message communication for a system. A job comprising a sequence of tasks and messages is formalized by timed automata based on a task and message model. Thus, the behaviour of an entire distributed system is a parallel composition of these timed automata models. Temporal logic properties are also formed to express constraints on the behaviour of the system components such as precedence relationships between tasks and messages. Schedules are automatically computed by formally verifying that these properties are satisfied with the given models. A successful schedule is simply generated by the verifier, in this case the UPPAAL model-checking tool. The work successfully demonstrates that the model

checking yields schedules for the difficult problems. This approach is flexible, tractable for small medium sized systems, and is extensible. However, there remain issues concerning scalability.

1.3 Summary of Contributions

The principal aim of this research is to find a more rigorous approach for schedule design in time-triggered architectures, in particular, for distributed real-time systems. By utilising timed automata functionality, a novel approach is proposed in this thesis, which is different from existing mathematical approaches. The proposed approach constructs timed automata models and temporal logic properties representing the behaviours and requirements of distributed real-time systems in a time-triggered architecture. If the property is satisfied with the given models, it is possible to produce schedules not only for local scheduling and but also for global scheduling for the systems. In this thesis, the following contributions are achieved:

- Proposal of a novel approach to designing schedules of distributed real-time systems in time-triggered architecture, particularly, using timed automata models and their functionality, which is entirely different from the existing mathematical approaches.
- Timed automata analysis to transform systems into timed automata models in order to apply the models to the proposed approach.
- Development of a prototype toolset based on the proposal approach in order to automatically generate schedules for the systems. In particular, the toolset provides an easy way to create the timed automata models and generate both local schedules and global schedules.
- Investigation of more constraints affecting the difficulty of designing schedules in time-triggered architecture and exploration of alternative timed automata models to overcome these constraints.

1.4 Organisation of the Thesis

This thesis comprises 8 further chapters as follows:

- Chapter 2 and Chapter 3 introduce the background of real-time scheduling theory and the background of timed automata respectively. The contents of these two chapters are the foundation of this thesis and methods adopted.
- Chapter 4 shows the way to transform systems into timed automata models. The chapter also includes the formal definitions of the terms used throughout this thesis to clarify the various definitions found in the literature.
- Chapter 5 explores different ways of constructing timed automata models in order to find an efficient way to express systems by timed automata models. Also, this stage introduces UPPAAL tool in more detail as the principal tool throughout this thesis.
- Chapter 6 introduces the development of a prototype toolset. Each step of the development is explained in full. Also this chapter includes evaluating the prototype toolset.
- Chapter 7 presents two case studies, an Adaptive Cruise Control system and a Robot Transport system. The structures and operations for the systems are described and the timed automata models for the system are applied to the approach. In addition, there are some experimental studies using the systems in order to find factors affecting the approach.
- Chapter 8 investigates more timing constraints that affect designing schedules in time-triggered architectures. Possible solutions for the constraints are discussed and introduced.
- Chapter 9 concludes the thesis and discusses future work.

Chapter 2

Background of Real-Time Scheduling Theory

Over the last few decades, considerable research has been undertaken in the areas of real-time scheduling and algorithms in order to improve schedulability of real-time systems such that all tasks meet their deadlines. This chapter begins by introducing types of tasks based on their arrival patterns. Following the introduction, it presents basic scheduling concepts with examples, depending on when scheduling decisions are made. There is also a debate issue between event-triggered and time-triggered architectures and the chapter ends by considering various non-optimal techniques.

2.1 Real-Time Scheduling Principle

Real-time systems, as mentioned previously, have to react within their timing requirements; the correct behaviour of these systems depends on not only the logical result of computation, but also on the time at which the results are produced [But97, BW01, Kop97, XP93]. In particular, a set of tasks in hard real-time systems are time-critical tasks and must meet their specified deadlines. When missing the deadlines, hard real-time systems may lead to catastrophic results such as the loss of human lives or assets. Thus, real-time scheduling is an extremely important activity in real-time systems, in order to guarantee that the timing requirements of real-time systems are met.

Real-time scheduling theory [SAÅ+04] provides a possible way to predict the timing behaviour of real-time systems even if tasks are concurrent with precedence constraints. There already exists a number of scheduling approaches and algorithms for predicting whether all tasks will always meet their deadlines. In general, depending on the characteristics of tasks in real-time systems such as whether tasks appear on regular

arrival pattern or not and whether tasks are preemptive or not, the way of analysing task scheduling is different. Moreover, depending on the timing requirements of tasks such as whether tasks require time-critical processing or not and whether tasks require an immediate response or not, scheduling can be applied in static or dynamic ways.

2.1.1 Types of Tasks

Based on the arrival pattern of tasks in real-time systems, three types of tasks are considered: *periodic*, *aperiodic* and *sporadic task.*

- A periodic task is a task which arrives at defined intervals. Thus all the instances of a periodic task in the future are known by adding multiples of its period. For example, the arrival time of i^{th} task t with period p is $t_i^p = (i-1)\times p$. It is easily realised that the schedule period of a set of tasks is the least common multiple of the periods of these tasks.
- An aperiodic task is a task which arrives randomly. Thus, it is impossible to predict the future arrival times of an aperiodic task; such a task can respond quickly to some external triggering events.
- A sporadic task is a task which also arrives randomly but at least the minimum interval time between consecutive instances of a sporadic task is known. Thus the scheduling of sporadic tasks is predicted by assuming their maximum arrival rate [CA95].

When considering hard real-time scheduling, an aperiodic task may have more chance of missing its deadline because of the difficulty of handling its uncertainty. Hence in most hard real-time scheduling it is assumed that the systems consist of period and sporadic tasks only. In some real-time scheduling, a task, while executing, may be pre-empted if more urgent tasks want to execute. Therefore, a task is further categorised into *a non-preemptive task* and *a preemptive task.*

- A non-preemptive task is completed without any interruption once it is started. This is useful for real-time systems which consider that many short tasks have to be executed.
- A preemptive task can be temporarily pre-empted during its execution by another task arrival according to a predefined scheduling policy such as a

higher-priority task invocation in priority driven scheduling. This is reasonable in real-time systems which require handling a more urgent task.

2.1.2 Classification of Scheduling

There are various existing scheduling policies for real-time systems. Depending on the time at which scheduling decisions are made, they can be typically categorised as *static* or *dynamic scheduling*, also often referred to as *off-line* or *on-line scheduling*.

- Static schedulers make their scheduling decisions statically prior to execution. With complete prior knowledge about all tasks such as maximum execution times, deadlines, precedence constraints, etc, *static scheduling* defines the entire schedule called a dispatching table. As it contains all information concerning when and which task is to be scheduled next at every point of a discrete time-base, the overhead of static scheduling at run time is small. *Static cyclic scheduling* is a well-known static scheduling policy [BW01, LA99]
- Dynamic schedulers make their scheduling decisions at run time, based on the priorities of task invocations. So the decisions have to take account of current and future availability of systems such as in cases that new tasks are created, or tasks' priorities need to be revaluated. Thus, *dynamic scheduling* is more complex and consumes more overheads than static scheduling. However, this scheduling policy has the benefit of being able to change the processing environment, and thus provides greater flexibility. Based on fixed and changeable priorities, dynamic scheduling can be further categorised into *fixed priority scheduling* and *dynamic priority scheduling* [ABD+95]. *Deadline monotonic scheduling* [ABR+91, Tin00] and *rate monotonic scheduling* [LL73] are well-known fixed priority scheduling policies, while *earliest deadline first* [But97] is a well-known dynamic priority scheduling policy.

2.1.2.1 Static Cyclic Scheduling

With a fixed set of periodic tasks, the static cyclic scheduling approach can provide an entirely deterministic schedule within the cyclical time interval in which all the periodic tasks have to execute at their correct rate [BW01, LA99]. With the deterministic schedule, it is possible to know which task is executing at any given time. So, there is

no actual work required at run time and also no need to protect a resource which is already managed by the schedule. It is expected that the cyclical time interval, which is called a major cycle for the schedule, consists of a number of minor cycles for each joining task. For example, if the major cycle is 100 time-units, a task having 20 time-units period will appear 5 times. Consider more tasks within 100 time-units such that there are 5 tasks A, B, C, D, E having the periods 25, 50, 50, 50, 100 and the execution times 10, 10, 5, 5, 5 respectively. One of deterministic schedules for these tasks may be expected as shown in Figure 2.1. According to the Figure, the task A arrives at time 0 and appears 4 times within 100 time-units due to its period known 25 time-units; the task B arrives at time 10 and 60 as its period 50 time-units; with 50 time-units of the period of the task C and D, the task C appears at time 20 and 70, and the task D is at time 40 and 90; finally the task E only arrives at time 45 as its period 100 time-units. Under this schedule within 100 time-units, there are no ways of missing the appearance of tasks with a purely fixed set of periodic tasks and thus this schedule is called a dispatching table.

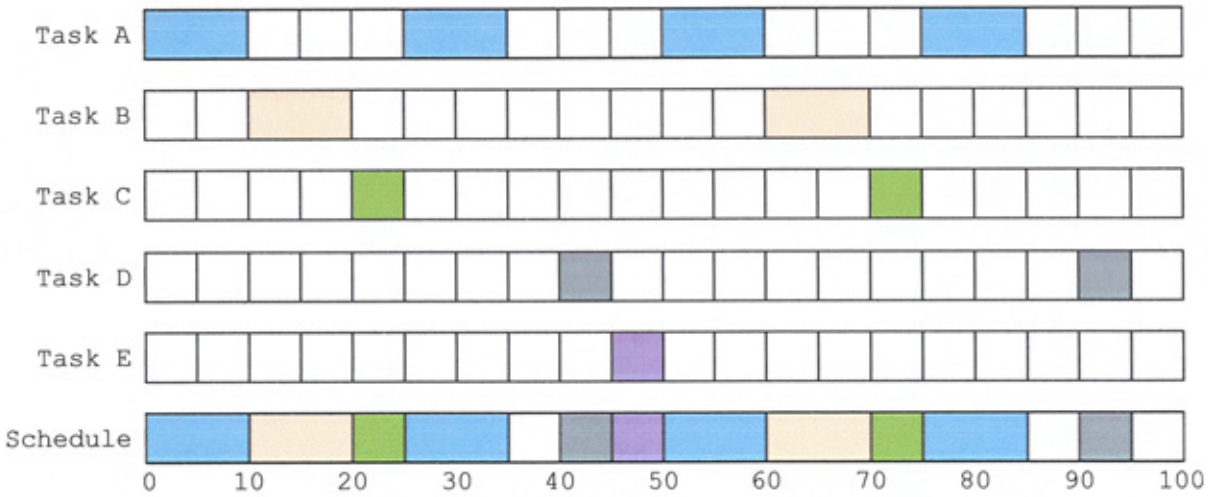


Figure 2.1: A Dispatching Table for Cyclic Schedule

Like this example, static cyclic scheduling is an effective approach for simple systems which have countable tasks. If it is likely to find a cyclic schedule for all tasks in systems, additional schedulability test may not need as the schedule is already proved [ATB93, XP00]. However, there are numerous drawbacks in static cyclic scheduling. As the principle of the scheduling is based on cyclic time intervals, it only supports

periodic tasks and needs to fit all tasks into the major cycle. Consequently, this makes the scheduling strict and difficult to construct. In particular, when considering large systems, finding a schedule for all tasks may be very time consuming work, known NP-hard [But97, TBW92] and the dispatch table may be very long, requiring considerable memory.

2.1.2.2 Fixed Priority Scheduling

Depending on whether tasks may be pre-empted or not, fixed priority scheduling can be divided: *Non-preemptive fixed priority scheduling* and *preemptive fixed priority scheduling*. The differences between the two scheduling are explained through a simple example. Suppose that there are 4 tasks F, G, H, I requiring 20, 20, 10, 30 time-units of computation times and with 45, 50, 40, 95 time-units of deadlines respectively; the priority of the tasks depends on their alphabet order; it means that the task F has the highest priory and the task I has the lowest priory among them.

When the F, G, H, I tasks arrive at time 45, 35, 10, 0 respectively, Figure 2.2 shows a possible execution sequence for these tasks under the non-preemptive fixed priority scheduling policy. The task I begins its execution first at time 0 and then the task H having a higher priority than the task I arrives at time 10. However, the task H must wait until the task I finishes due to the non-preemptive policy even though it has a higher priority. Thus the task H is blocked until 30 time-units and starts after the time. Similarly, the task G arrives at time 35 while the task H is currently executing and thus it is also blocked for 5 time-units and starts at time 40. The highest priority, task F, also is blocked by the task G for 30 time-units and starts at time 60 when the task F finishes its execution. Under the non-preemptive policy, the tasks H, G and F are blocked by their previous tasks although they have a higher priority than the tasks currently executing when they arrive. However, even with these blocked times during their execution times there is no problem in meeting their deadlines in this scenario.

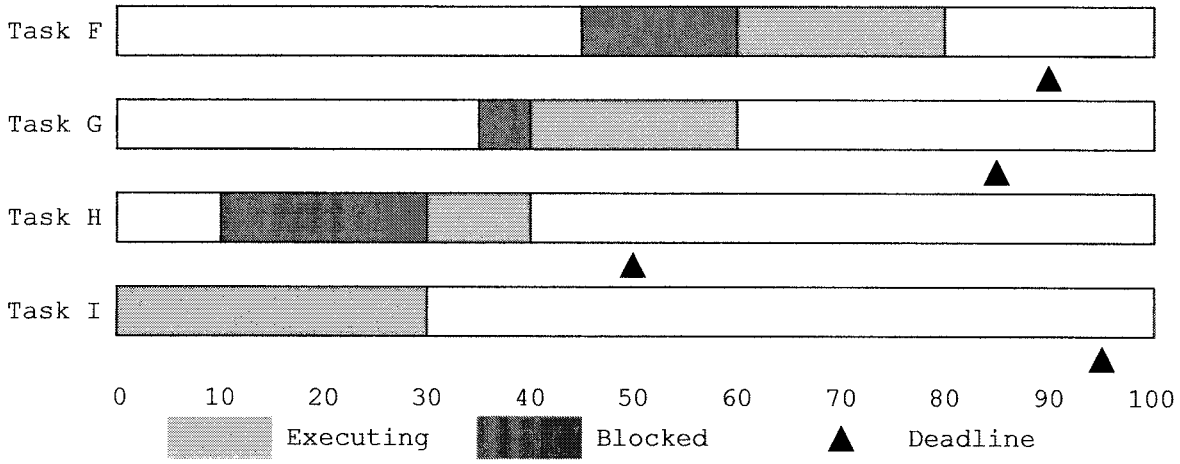


Figure 2.2: Non-Preemptive Fixed Priority Schedule

The non-preemptive fixed priority scheduling can be easily analysed for schedulability as a task can execute without interruption. It is obvious that when a task begins its executing without blocking, the finishing time is the same as its execution time. Even though a task is blocked, the finishing time is the sum of its blocking time and execution time. However, problems of lack of responsiveness are raised as the drawback of the scheduling policy. As seen in Figure 2.2, the higher priority tasks are blocked by the lower priority tasks. This means that the higher priority tasks, which are usually allocated for an emergency in the environment, will suffer from blocking.

Under the preemptive fixed priory scheduling, Figure 2.3 shows how these four tasks are executed. The task I is again a first task to begin its execution and then the task H having a higher priority than the task I arrives at time 10. This time the task H can execute immediately as it has a higher priority, while the task I is blocked until there are no further higher priority tasks available to execute. When the task H finishes at time 20, the task I resumes its execution due to no higher priority tasks available. At time 35, the task I is again preempted by a higher priority, the task G, and then at time 45, the task G is also preempted by the highest priority task F. When the task F finishes at time 65, the task G and I are waiting to execute but as the task G is a higher priority than the task I, the task G resumes its execution while the task I is still being blocked. The task I eventually resumes its execution at time 75 and finishes its execution at time 80. Under the preemptive policy, the task I has the least benefit as it is blocked by the task F, G and H having the higher priority. However, the task F and H benefit from the preemptive

policy by preempting other tasks and then executing immediately. Nevertheless, all the tasks still meet their deadlines.

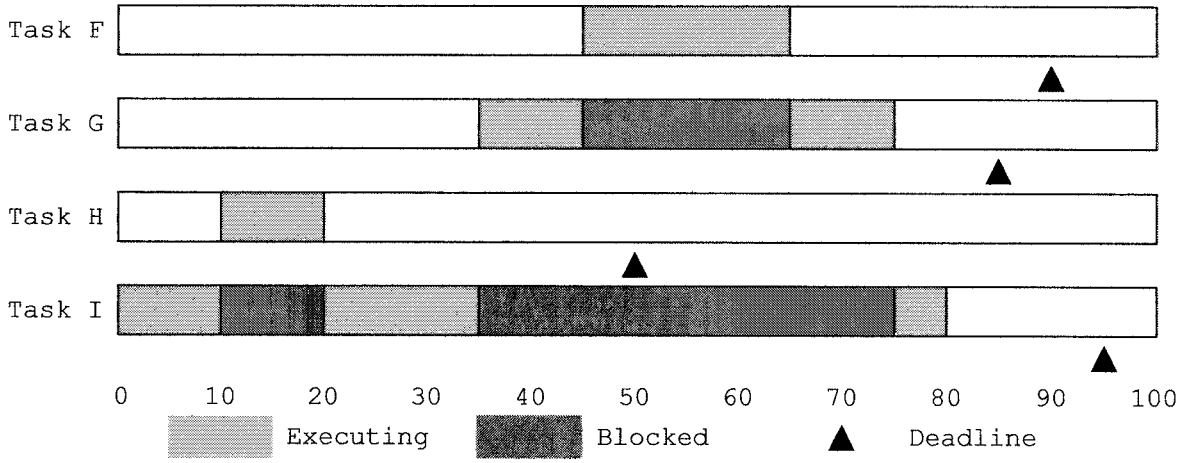


Figure 2.3: Preemptive Fixed Priority Schedule

The preemptive fixed priority scheduling provides faster responsiveness to higher-priority tasks, compared the non-preemptive fixed priority scheduling although lower-priority tasks suffer from additional blocking. At least under the preemptive scheduling policy, higher priority tasks do not have much chance to miss their deadlines. With this certainty of quick response for higher-priority tasks, numerous commercial real-time operating systems are available to support preemptive scheduling, in particular, fixed priority scheduling. However this scheduling policy also has drawbacks such as complex systems suffer frequent preemption among tasks, the systems may suffer from a context switching overhead required a significant time.

2.1.2.3 Dynamic Priority Scheduling

In dynamic priority scheduling, tasks do not have allocated priorities but the priorities is decided by the circumstance of tasks. Earliest deadline first scheduling is widely used. In this scheduling, tasks are executed in the order determined by the absolute deadlines of tasks and thus a task which has the shortest (or nearest) deadline will be given the highest priority. Applying the earliest deadline first scheduling to the same example described above, Figure 2.4 shows how the scheduling is worked. When the task H arrives at time 10, it preempts the task I because the task H has a shorter deadline, 50

time-units, than the task I, 95 time-units. When the task F arrives at time 45, it cannot preempts the task G as the task F is not the shortest deadline task, 90 time-units, than the task G , 85 time units and thus the task F is blocked. However, the task F can run at time 55 after the task G finishes. This time the task F having its deadline, 90 time-units, beats the deadline of the task I, 95 time-units.

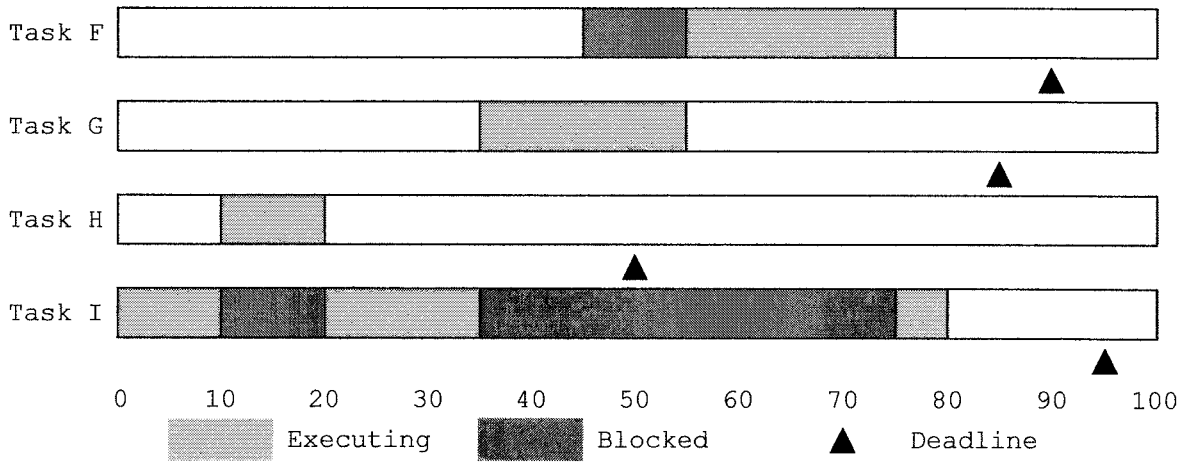


Figure 2.4: Preemptive Dynamic Priority Schedule

This scheduling provides an ideal way to guarantee that as many as possible of the tasks are able to meet their deadline, giving the highest priority to the most urgent task among the waiting tasks. However, it suffers from the overhead for recalculating the priorities of all the tasks again whenever a task arrives and finishes.

2.1.2.4 Comparison between Fixed Priority and Static Cyclic Scheduling

An interesting study of Lönn *et al.* al [LA99] compares the timing characteristics when fixed priority and static cyclic scheduling are used, in particular, with global time or not. Using an example of a simple control loop, they analyse the effects of the expected response time to the control loop and also further study the effects of using a global time base. They assume that the control loop is implemented with two computers connected via a communication bus; the control loop collects a sensor input from a sample computer and transmits the input data over the bus to an actuator computer. In the loop, they have combined static cyclic and fixed priority scheduling on processors,

and static cyclic and fixed priority scheduling on a communication bus. For each combination, the results are given using global time or not.

According to the results of their work, it appears that the combination of static cyclic scheduling on a processor and static cyclic scheduling on a communication bus using a global time will give the smallest control delay. The global time synchronised local clocks in each processor can provide fixed offset values to the sensor and actuator tasks in the control loop and thus it reduces the significant variation in control delay; without global time these tasks have to be synchronised from sensing to actuating by message passing via the communication bus. Tasks in the fixed priory scheduling suffer release jitter, and blocking and interference from other tasks, however using a global time which provides offset values will improve the response time even though it is complicated. They conclude that tasks requiring strict requirement such as small timing variation will be more efficiently implemented in static cyclic schedule systems, while tasks requiring immediate response, such as event handling, will be more efficiently handled in fixed priority scheduled systems.

2.1.3 Schedulability Analysis

A schedulability analysis can help to accurately predict whether a set of tasks is scheduled or not in real-time systems and thus is able to decide the failure of the systems. Depending on the accuracy of the analysis, it is categorised to *exact*, *necessary* and *sufficient* schedulability analysis. For example, if a system is positive in sufficient schedulability analysis, the tasks in a system will be definitely schedulable. If necessary analysis is negative, the tasks will not be schedulable. There are a number of schedulability analyses in the literature; typically they are based on *processor utilisation* and *response time analysis* [BW01, But97]. These analyses are summarised below.

2.1.3.1 Processor Utilization Analysis

With a rate monotonic scheduling, i.e. tasks having shorter periods are granted higher priorities, a simple processor utilization analysis is intuitively known that if the sum of all tasks' processor required time (the required time of each task is computed by dividing its computation time with its period time) is less or equal to the capacity of a processor (100%), then the tasks are assumed to be schedulable.

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \qquad (1)$$

Consider the following set of tasks in a processor, task J, K and L having the period 20, 40, 60 time-units and computation time 5, 10, 30 time-units (See Table 2.1). According to the processor utilization analysis, this task set is schedulable as the utilization of the tasks, $\frac{5}{20}+\frac{10}{40}+\frac{30}{60}=1$, is the same as the capacity of the processor.

Task	Period	Computation Time
Task J	20	5
Task K	40	10
Task L	60	30

Table 2.1: Task Set A

However, this schedulability analysis fails because the Task L having the lowest priority misses its period 60 and finishes its computation at 70 time-units over its period, shown in Figure 2.5. This is because the Task L often suffers from the preemption of higher priority tasks. Although the Task Set A fails, it works in a case that the task periods are multiple of the highest priority task period. For instance, if the Task L has the period 80 and computation time 40, the set of tasks is schedulable, shown in Figure 2.6.

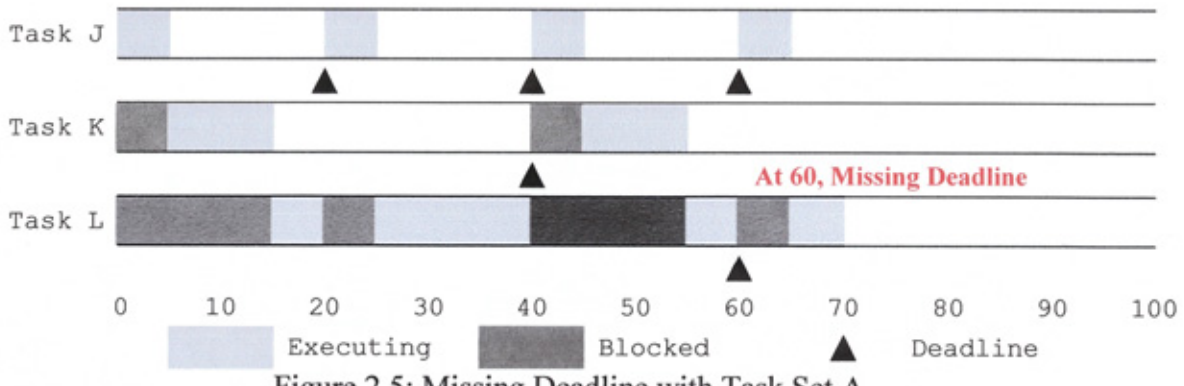


Figure 2.5: Missing Deadline with Task Set A

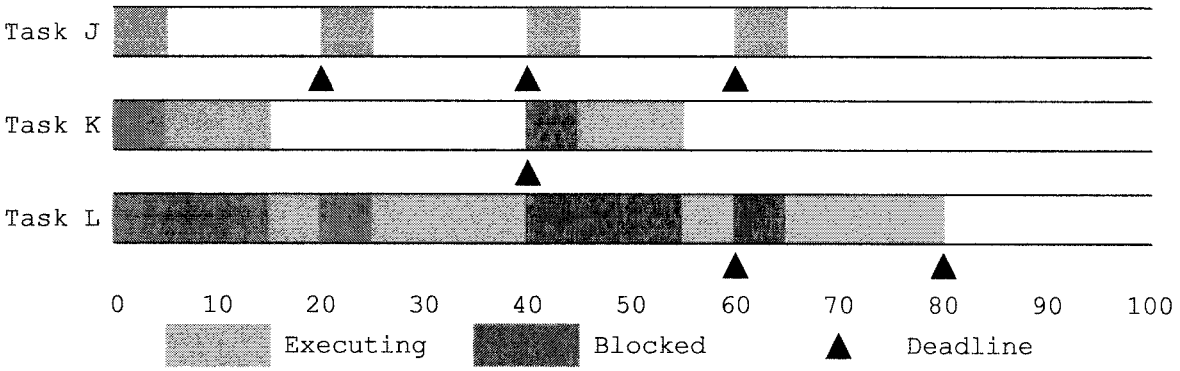


Figure 2.6: Not Missing Deadline with Task Set A

Liu and Layland [LL73] introduced a schedulability analysis which considers the utilization of tasks based on the rate monotonic scheduling. They propose *processor utilization bound* depending on the number of tasks rather than the constant utilization value, 1.

$$\sum_{i=1}^{N}\left(\frac{C_i}{T_i}\right) \leq N(2^{\frac{1}{N}} - 1) \qquad (2)$$

If this condition is satisfied with N tasks, all the tasks will meet their deadlines. For instance, if the task set A wants to be schedulable, the total utilization of the task set should not exceed 0.78 ($3(2^{\frac{1}{3}}-1)$). According to (2), the utilization bound values are listed below as percentage.

Task	Utilization bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

Table 2.2: Utilization Bounds

When the number of tasks N is getting increased, the utilization bound is closing to 69.3%. This means that any task set whose combined utilization is less than 69.3% will always be schedulable on a preemptive priority-based scheduling with priorities assigned by the rate monotonic scheduling.

However, the Figure 2.6 shows that the task set A is schedulable even though the set exceeds the utilization bound. This means that this analysis is sufficient but not necessary. Consequently, if a task set passes this analysis, the set will meet all its deadlines; otherwise it may or may not be schedulable [Kop97].

2.1.3.2 Response Time Analysis

The processor utilization analysis described above has numerous drawbacks. It provides yes/no answer but does not provide any detail of the actual response time for each task; the analysis is not generally applicable for various task sets; it is a sufficient analysis but not a necessary analysis. In contrast to the processor utilization analysis, the response time schedulability analysis, however, is applicable to any task set, which is based on fixed priority and preemptive. In particular, this analysis can predict the worst-case response time of each task. If the worst-case response time R_i for each invocation of a task i is less than the deadline D_i, the task is schedulable, i.e. $R_i \leq D_i$.

In the response time schedulability analysis, the worst-case response time of the highest priority task will be equal its own computation time intuitively, i.e. $R_i = C_i$ as there are no disturbances. The worst-case response time of task i, which is one of the remaining tasks except for the highest, will be its own worst-case computation C_i plus the interference I_i from higher-priority tasks, i.e. $R_i = C_i + I_i$. The maximum interference (3) is occurred when all higher-priority tasks are released at the same time as the task i. It is calculated as multiplying the computation time of each higher-priority task j by the number of releases of higher-priority task j, R_i, in the period of task j, T_j. Thus I_i is denoted by

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{3}$$

where $hp(i)$ is the set of higher-priority tasks than the task i and $\lceil x \rceil$ is an ceiling function to compute the number of release time, which gives the smallest integer number of x. For example, 1/4 is 1, 4/4 is 1 and 5/4 is 2. Then, this interference is applied to R_i as (4).

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{4}$$

However, R_i appears on both sides of the equation as R_i is being used in the calculation. The simple way of solving this problem is to calculate the response time of the task i iteratively [ABR+93].

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{5}$$

The amended equation (5) suggests that only when $R_i^n = R_i^{n+1}$, the response time of R_i can be found. The calculation starts by R_i^0 and then the value of R_i^0 is employed in R_i^1. This will be continued until $R_i^n = R_i^{n+1}$. Whenever R_i^n converges to a value and it is not greater than the period of the task i, T_i, it is said that the task i is schedulable. On the other hand, the task i is not schedulable if R_i^n exceeds T_i.

Task	Period	Computation Time	Deadline	Priority
Task M	20	5	10	High
Task N	30	10	20	Medium
Task O	90	30	50	Low

Table 2.3: Task Set B

Consider the task set B. The response time of `Task M` as the highest-priority task is the same as its computation time, 5. Next, the worst-case response time of `Task N` is as follows: R_N^0 is 10; and then R_N^1 is 15 with R_N^0 interfered by the `Task M`; eventually the worst-case response time R_N is 15 as $R_N^1 = R_N^2$. See this calculation below:

$$R_N^0 = 10$$
$$R_N^1 = 10 + \left\lceil \frac{10}{20} \right\rceil 5 = 15$$
$$R_N^2 = 10 + \left\lceil \frac{15}{20} \right\rceil 5 = 15$$

The worst-case response time for the lower-priority `Task O` converges to 80 because $R_O^3 = R_O^4$. The calculation is as follows:

$$R_O^0 = 30$$

$$R_O^1 = 30 + \left\lceil \frac{30}{20} \right\rceil 5 + \left\lceil \frac{30}{30} \right\rceil 10 = 50$$

$$R_O^2 = 30 + \left\lceil \frac{50}{20} \right\rceil 5 + \left\lceil \frac{50}{30} \right\rceil 10 = 65$$

$$R_O^3 = 30 + \left\lceil \frac{65}{20} \right\rceil 5 + \left\lceil \frac{65}{30} \right\rceil 10 = 80$$

$$R_O^4 = 30 + \left\lceil \frac{80}{20} \right\rceil 5 + \left\lceil \frac{80}{30} \right\rceil 10 = 80$$

According to the response time schedulability analysis, all the tasks in the task set B are schedulable. Figure 2.7 shows the worst-case behaviour of this task set and indeed, it proves that they are schedulable as indicated by the schedulability analysis.

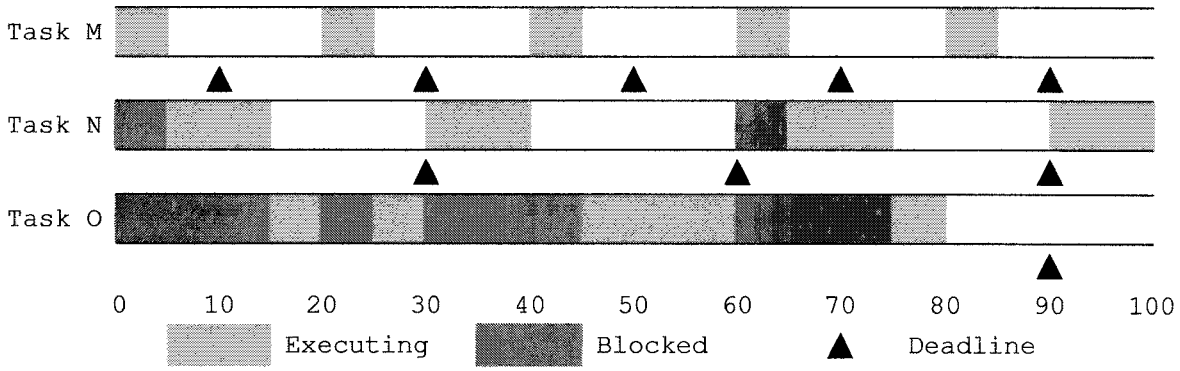


Figure 2.7: The Schedulability Analysis for the Task Set B

2.1.3.3 Schedulability Analysis in Distributed Real-Time Systems

Although numerous approaches have been successfully applied to single processor architecture, schedulability analysis for distributed real-time systems has received less attention and is much less complete. One of well-known schedulability analysis for distributed systems is the '*holistic*' approach which is introduced by Tindell *et al.* [TC94]. They introduce the analysis of distributed hard real-time systems. In particular, this analysis focuses on systems with simple fixed priority scheduling and a simple time division multiple access protocol. Basically, Tindell *et al.* [TC94] apply schedulability analysis to fixed priority tasks on each processor, and then extend it to messages, in order to determine the worst-case response times of messages sent between processors. Consequently this analysis can address the delivery costs of messages both the

overheads on a destination processor and the delivery times of the message. The purpose of this analysis is not just to give a priori schedulability guarantees, but also to aid the configuration of such a distributed system.

Pop *et al.* [PEP99a, Pop00, Pop03] present a schedulability analysis that is similar to Tindell *et al.* However, it adopts time triggered protocol (TTP), mainly because of reasons of fault tolerance introduced using TTP [Kop97]. The response times calculated using the schedulability analysis, under a given task and message set, are combined in a cost function that measures the degree of schedulability. The outcome of the schedulability analysis is a message descriptor list (MEDL), which presents a global schedule, and a message handling time table (MHTP), which is, a local schedule. In addition, Pop *et al.* [PEP99a] compare four different approaches – static single message allocation (SM), static multiple message allocation (MM), dynamic message allocation (DM), and Dynamic Packet Allocation (DP) – over TTP. Thus, Pop *et al.* contribute the result that it is not only possible to determine if a certain task set implemented on a TTP-based distributed architecture is schedulable, but it is also possible to select a particular message passing strategy and also to optimise certain parameters of the communication.

A schedulability analysis for controller area network (CAN) is performed by Tindell *et al.* [TBW91]. CAN is a well-designed communications protocol for sending and receiving short real-time control message. CAN is designed for use in automobile industry, and is very popular in this environment. Tindell *et al.* [TBW91] apply the analysis to fixed priority preemptive real-time processor scheduling on CAN bus in order to bound the response times of messages. A holistic approach to performance prediction of distributed real-time CAN systems is applied by Henderson *et al.* [HKR98].

2.2 Event-Triggered and Time-Triggered Architecture

When considering the design of real-time systems, there are at present two fundamentally different designing architectures. An event-triggered architecture is driven by the occurrences of events and not by the passage of time, while a time-triggered architecture is driven by a recurring clock tick and the pre-determined points

in time. Depending on which architecture is chosen during the design, it affects the entire system such as scheduling decisions, communication protocols, occurrence of an application task, structure of hardware, operating systems, etc. Thus it is an important assignment to choose a right one. It is generally known that an event-triggered architecture is well-fitted in non-safety critical systems and a time-triggered architecture in safety critical systems. Here, there are summaries of the different characteristic attributes between two architectures, such as predictability, resource utilization, extensibility and testability, based on the researches performed by Kopetz [Kop91, Kop93, Kop95, Kop98].

2.2.1 Predictability

The predictability is one of important factors in real-time systems as providing to meet the deadlines imposed by its environment and thus it prevents the failure of systems. Because a time-triggered architecture is based on the pre-determined points in time, the allocation plans for tasks on a processor and messages on a communication network are required during the design. With the detailed plans for the temporal behaviour of each task and message, the behaviour of the system on the architecture can be predicted precisely.

By contrast with a time-triggered architecture, an event-triggered architecture does not require the creation of a set of detailed plans during the design as the execution for tasks and messages on a system are determined dynamically. Depending on the specific application scenarios in a system, different schedules are unfolded dynamically to meet its timeliness requirements. Thus, in an event-triggered architecture it is hard to do analytical schedulability and also difficult to predict the behaviour of a system on the architecture precisely.

2.2.2 Resource Utilization

As all schedules in a time-triggered architecture are already fixed and planned whether a system is under the maximum load or not, the resource utilization will be always known and can be pre-determined. The architecture might have the different utilizations only if many different operating modes have to be considered. Creating well-defined schedules on the architecture can improve resource utilization in spite of the difficulty in finding

the schedules as suffering from the explosion problem, known NP-hard problem [TBW92, But97].

An event-triggered architecture only has to schedule the tasks which have been activated under actual circumstance. So, the resource utilization of the architecture varies soon after the tasks finish their execution times. However, the architecture additionally requires run-time resources such as the execution for the dynamic scheduling algorithm, the synchronization, the buffer management, the interrupt handling, etc.

Depending on the load conditions of a system such as how many tasks are loaded on the system, the architectures are competitive. If the load conditions are low or average, the event-triggered architecture will have better performance than the time-triggered. If a system is considered on the peak load conditions, the time-triggered architecture is the best choice.

2.2.3 Extensibility

Whenever occurring to change the existing functions or to add the new functions into an existing system, it is perceived how much the extensibility is important to successful systems in future. In an event-triggered architecture it is easy to change an operative task and to add a new task into a system as all the functions on the system are determined at run time.

In a time-triggered architecture, the extensibility depends on the temporal time which has been already allocated to each task and each message on a system. In a small change such as a modifying task does not exceed its allocation time, the change will not have temporal effects to the rest of the system. However, if a modifying task exceeds the allocation time or a new task is added, the existing static schedules for the system have to be recalculated. Moreover, adding a new processor on distributed systems may suffer from overhead for the recalculation of communication schedules if the processor sends a message to the existing system.

It is obvious that an event-triggered architecture has better extensibility than a time-triggered architecture. In conclusion, if a system often requires extending and changing tasks, an event-triggered architecture is a better solution in order to avoid a recalculation

of the static schedule in a time-triggered architecture. However, the increasing changes on the system cause a burden to retest the whole system.

2.2.4 Testability

As all tasks in a time-triggered architecture are pre-determined in time-base, every input value from the tasks can be observed and reproduced. Thus, the overall testing for a system can be performed with the established detailed plans and be well-constructive. When considering, in particular, the critical issues on a system, this architecture will give a good answer with confidence.

A system testing in an event-triggered architecture has a difficulty to observe and reproduce all the events from the tasks which has not been decided yet. The testing for the system is usually based on a simulated mode. However, it is still not sufficient when considering the system performance in the peak load. Thus the overall testing for a system will be less established with the detailed plans and less constructive. It will be also difficult to get a confidence answer without the established detailed plans on critical systems. Because of the reasons of differences between two architectures, the effort for testing a system in an event-triggered architecture, thus, is much greater than that for the testing of the corresponding system in a time-triggered architecture.

2.3 Global Scheduling

The term "global scheduling" is considered here as scheduling of tasks in a processor and also the scheduling of communications between tasks that are allocated to different processors. Global scheduling generation for multiprocessor systems is problematic and is known to be NP-hard. Thus, it is necessary to find algorithms which simplify the problem and give feasible solutions. There are a number of existing algorithms for solving the scheduling problem in the literature, e.g. Burns *et al.* [BHR95] explain implementation of feasible cyclic schedules for a number of algorithms. The following scheduling algorithms are widely known in the literature: *heuristic approach*, *simulated annealing*, *genetic algorithm* and *tabu search*. Each of these algorithms is briefly described below.

2.3.1 Heuristic Approach

Due to their ability to easily find feasible solutions, various heuristic approaches are commonly adopted. A heuristic approach basically uses some rules defined by the designer based on intuition, experience, etc. such as the shortest period task is allocated first in scheduling allocation. This means that it is not based on the absolute accuracy but on an approximation. The approach may find a feasible solution among all possible solutions but does not guarantee that the solution is the best. However, heuristic approaches can lead to solutions which are good enough and they may even find a solution closed to optimal one. Two such algorithms are now described.

2.3.1.1 List Scheduling

List scheduling algorithm [LAA+94] is one of the classic heuristic scheduling techniques developed in the operations research community. The operation principle of this algorithm is to make an ordered list of ready tasks depending on their priories which are determined statically before scheduling processes begins, and then find the most suitable available processor for each task picked from the list until the list is empty. Figure 2.8 shows the simple pseudo code for the algorithm where L denotes the ordered list of ready tasks, $\mathcal{P}$ is a set of processors and c_P is the available time of the processor P. This example does not consider preemption.

```
Each task is assigned a priority
Create a task list L ordered by priority
Initiate processors P

do while L is not empty
    P = Get the most available processor from P
    c_P = Get an available time from P
    t_i = first task in L
    allocate t_i to processor P at time c_P
end do
```

Figure 2.8: List Scheduling Algorithm

The algorithm can be varied depending on the way that tasks are assigned priorities and the way that the most suitable processor is chosen. One of priority functions for assigning the priorities of the tasks is based on the earliest deadline first (EDF) which is

already described above. The EDF is optimal to allocate the priorities for a single processor but is not for systems with multiprocessors [BLM+98, Red98].

2.3.1.2 Iterative Deepening A*

A* [DP85] is a graph/tree search algorithm which finds a path from a given initial node to a given goal node. It employs a heuristic estimate to find the best route, and visits the nodes in order of this heuristic estimate. The Iterative Deepening A* (IDA*) algorithm [Kor85] is a derived version of A* with the same properties as A*, such as optimality and completeness but reduces use of storage space. IDA* performs iteratively depth first searches with successively increased cost-thresholds in order to traverse paths as long as a given threshold is not exceeded. At each iteration, IDA* does the search and removes all nodes exceeding the threshold. Then, the threshold is increased to the minimum path value that exceeds the previous threshold; then the process is repeated until a path is found. It employs a heuristic function $f(n)$ that estimates the cost of each path and thus omits useless paths and follow promising paths. The heuristic function $f(n)$ generally consists of $g(n)$, the cost already spent in reaching to n, and $h(n)$, the estimated cost of the path from n to a goal node. By adding these, $g(n) + h(n)$, it is possible to derive the estimated cost from the root node to some goal nodes.

IDA* was applied in the MARS project [KM85] in order to find feasible schedules. It is based on the assumptions that every node in the algorithm represents the decision to schedule a given set of tasks or messages at a given point time; every path from the root node to the goal node represents a complete feasible schedule. MARS is a fault-tolerant distributed real-time system where each processor works on a cycle schedule and each transaction is strictly time driven and periodic on a TDMA protocol. It is aimed to maintain a deterministic behaviour even under peak load conditions. However, its strict period prevents its general application. For the MARS project, Fohler *et al.* [FK90] implemented the IDA* algorithm with some modifications; they focused on reducing the run-time of the algorithm rather than finding optimal schedules. Figure 2.9 shows the overview of the IDA* algorithm.

```
ITERATION()
{
    if STARTTHRESHOLD is nothing then
        threshold = f(root)
    else
        threshold = STARTTHRESHOLD
    end if
    do while solution is found
        DEEPENING(root)
        threshold = threshold + min_exceed
    end do
}

DEEPENING( n )
{
    if solution is found with n then exit
    S = Create all successors of n
    N = Sort all nodes in S using f(n), leading by the best node
    do while the count of N is less than BRANCHINGFACTOR
        bn = Select the first node from N
        if f(bn) < threshold then
            DEEPENING( bn )
        else
            min_exceed = min(min_exceed, f(bn) )
        end if
        Remove bn from N
    end do
}
```

Figure 2.9: IDA* Algorithm used in MARS project

The algorithm starts by calling the ITERATION function which makes another function call to the DEEPENING function every a single iteration. The DEEPENING function actually performs the search by iteratively calling itself as long as a threshold is not exceeded. In comparison with a pure IDA*, the algorithm shown in Figure 2.9 additionally includes two parameters, STARTTHRESHOLD and BRANCHINGFACTOR for the following reasons:

STARTTHRESHOLD: IDA* initially uses the threshold, *f(root)*, in order to avoid loosing any solution, thus guaranteeing to find a (optimal) solution. However, the algorithm may start with higher values of the threshold, STARTTHRESHOLD, as it focuses on constructing a single schedule which guarantees all timing requirement to be met.

BRANCHINGFACTOR: It is possible to control the number of successors expanded, i.e. restring the size of search, using BRANCHINGFACTOR. It is caused that the algorithm

includes sort of successors according to their heuristic estimate and thus provides the most promising successor first. This factor can improve the run-time of the algorithm obviously but reduce the chances of finding a solution if this parameter is assigned to too small.

The heuristic function, $f(n)$, in the algorithm estimates the value of the time needed to complete the execution of each transaction in a system, called *time until response* (TUR). There are three factors influencing the calculation of TUR: maximal execution times for tasks, sum of the communication times and idles times in a transaction. In particular, it is important to estimate the sum of the communication times because it depends on the execution times of tasks and the availability of communication slots. Also, the estimation of the communication times will depend on the number of waiting messages and the availability of the time slot because the TDMA protocol used in the MARS project provides only one slot for each component.

2.3.2 Simulated Annealing

The simulated annealing algorithm was developed to observe how molten solid crystallises, involving heating and cooling of the solid to increase the size of its crystals and reduce its defect. The idea of the algorithm is to use the analogy between a combinatorial optimisation problem and the annealing processes of molten solid and it has, in particular, the ability of random search to potential new solutions. In the algorithm, the states of solid represent a feasible solution; the values computed by the states correspond to an energy of the solutions; the states which have a minimum energy correspond to the optimal solution to the optimisation problem [Red98, PK98].

The algorithm generally includes the following procedures: when given an initial solution, a neighbour solution is newly selected from the initial solution. Then, the energy function computes an energy of the new solution whether the energy is acceptable or not. In a case that it is acceptable such as the energy is less than the current energy, the energy is recorded as a best energy and the neighbour is promoted newly to a starting solution. These procedures are iterated until the optimal solution is found. Figure 2.10 shows the overview of simulated annealing algorithm. In particular, the algorithm shown in the Figure includes another way of changing a neighbour

solution to a starting solution such as in a case of $random(0,1) < e(E,E_s,T)$ where $random(0,1)$ is a random number generator between 0 and 1, and T is a temperature which is slowly reduced the annealing process. This is introduced for randomness in the algorithm and reduces any possibility that the algorithm discovers only local minima. For example, the algorithm does not find a better energy in a certain point and so it does not move to another starting solution.

```
Choose an initial temperature T and an initial solution S
E = Energy of solution S

do while criterion (E = 0) is not met
    N = a new solution from a neighbour of S
    E_s = Energy of solution N
    if E_s is acceptable (E_s < E) then
        E = E_s ; S = N
    else
        if N is a new staring solution (random(0,1) < e(E,E_s,T)) then
            S = N
        end if
    end if
    if T is required to change then
        T = new temperature lower than T
    end if
end do
```

Figure 2.10: Simulated Annealing Algorithm

As the simulated annealing algorithm can deal with highly nonlinear models with many constraints as its flexibility and ability to global optimisation problem, it is often applied to solve global scheduling problems such as the research works by [BNT+93, TBW91, TBW92]. In such algorithms, all tasks are randomly allocated and energy function is the sum of all missed timing constraints such as deadlines, latencies, communication delays, etc. In particular, finding a neighbour solution from the existing solution is an important part in the algorithm for scheduling.

2.3.3 Genetic Algorithm

A genetic algorithm is a search technique to find solutions to optimisation and search problems in complex multi-dimensional search spaces. It is developed by Holland

[Hol75] based on the natural evolution such that strong creatures in a species will live, whereas the weaker creatures will die off. And so, the species will gradually become stronger from generation to generation and be able to adapt its changing environment. For this reason, the operators used in the algorithm include crossover, mutation and natural selection, known as genetic operators.

```
Generate an initial population P = {ρ1, ρ2, ..., ρn}
Compute E(ρn) for all ρ in P

do while criterion (E(ρn) = 0 in P) is not met
    N = the n best solutions of P
    Perform crossover(N) and mutation(N)
    P = N
    Compute E(ρn) for all ρ in P
end do
```

Figure 2.11: Genetic Algorithm

A genetic algorithm starts with creating an initial population which is the set of possible solution of the optimisation problem. Each solution of the population, called an individual, is required to evaluate its cost by a cost function; it is understood that the individual having a lower cost is the fitter solution. Based on their costs, the algorithm selects the n number of best individuals from the population; once the best individuals are selected, the algorithm executes some operations in order to create a new population such as crossover which is used to create two new individuals from two existing individuals picked from the current population by the selection operation, and mutation which is used to create one individual from the one existing individual. The genetic algorithm iterates these procedures until some stopping criterion is met. Figure 2.11 shows a simple overview of the algorithm.

Considering the genetic algorithm in scheduling views, it requires a better way of representing an individual such as arbitrary data structure rather than bit-string which was originally used. And also, there is difficult to create appropriate genetic operators for scheduling and so the algorithm prefers to include mutation rather than crossover because mutation is easier to create. In addition, as it is important to create an initial

population in order to find an optimal solution in less time, the algorithm is recommend for creating the initial population based on the a priori knowledge rather than randomly.

2.3.4 Tabu Search

The Tabu search algorithm was developed by Glover et al. [GTW93] for solving combinatorial optimisation problems; in particular, the algorithm uses a flexible memory and so it is able to eliminate local minima and to search global areas. Although there is a similarity between the tabu search and a simulated annealing which are based on searching a neighbour solution, the tabu search generates the set of neighbour solution rather than one.

```
Create an initial solution S and a memory M
S'' = S
do while stopping condition (E(S'') = 0) is not met
    C = a candidate list of S using M
    S' = the best of in C
    S = S'
    if E(S') < E(S'') then
        S'' = S'
    end if
    Update M with C
end do
```

Figure 2.12: Tabu Search Algorithm

With an initial solution, the algorithm generates a possible candidate list of the solution; then it evaluates each solution in the list and compares to find the best solution among them. If the best solution is not an optimal solution, the algorithm generates a new list based on the best solution and evaluates it again until the optimal solution is found. However, there is an important element of the tabu search – the algorithm remembers all the solutions that have already been examined and does not allow them to be included in the list again. Thus, it is possible to eliminate local minima. Figure 2.12 explains the brief overview of the tabu search algorithm.

Because tabu search can use a memory, it has a more deterministic ability than random algorithms such as genetic algorithm and simulated annealing. However,

because of this, the algorithm may require impractically large memory. Thus, it is necessary to employ good strategies when selecting a list using the memory such as forbidding strategy, freeing strategy and short-term strategy [Glo89, Glo90].

2.4 Summary

This chapter has introduced the background of real-time systems. First it included types of tasks depending on their arrival patterns and on their preemption. In particular, it presented various existing scheduling policies for real-time systems, depending on the time at which scheduling decision are made. According to the scheduling policies, static cyclic scheduling is an effective approach for simple systems which have countable tasks even though it is strict and difficult to construct its static schedule; preemptive fixed priority scheduling has faster responsiveness to higher-priority tasks, compared the non-preemptive fixed priority; dynamic priority scheduling provides an ideal way to guarantee the most urgent task among the waiting tasks in a system but it suffers from the overhead for recalculating the priorities of the tasks. A schedulability analysis also has been demonstrated, considering processor utilisation and response time for real-time systems. The processor utilization analysis is easy to determine whether or not a task set in a system is schedulable but it does not provide any detail of the actual response time for each task, while the response time analysis can predict the worst-case response time of each task and is applicable to various task sets.

There has been a comparison between event-triggered and time-triggered architecture. A time-triggered architecture can provide accurate prediction and effective resource utilisation even if a system is under the maximum load or not, and so it is recommended to a system which requires more reliability and safety. An event-triggered architecture has immediate response and is easily extensible because of its dynamism, and consequently it is fitted to a system which requires a faster response and often extension.

There are a number of different algorithms for global scheduling found in the literature, including heuristic approaches, simulated annealing, genetic algorithm, tabu search etc and here each algorithm has been explained briefly. As they are based on a heuristic approach, they can find a non-optimal solution fast and easily but lead the

solution to be good enough. In spite of such this problem, they are widely accepted to solve global optimisation problems.

Chapter 3

Background of Timed Automata

This chapter presents the background of timed automata fundamental to the approach to designing schedules of distributed real-time systems in a time-triggered architecture. The properties of timed automata and their definition proposed by Alur and Dill [AD90, AD94] are reviewed, and followed by the techniques to reduce the state explosion problem inherent in timed automata. Also there is an introduction to temporal logic employed as a formal specification of timed automaton's requirements.

3.1 Timed Automata

Model checking employs automata to model systems in order to verify their correctness. *Automata* are the basis of the operational models used to specify the behaviour of a system which has to be validated; an *automaton* is a machine evolving from one state to another state by the action of transitions [BBF+01]. A state and transition are typically depicted by a circle and arrow respectively.

Timed automata first appeared in [AD90, AD94] as finite state automata, in particular, with a set of real-valued variables called *clocks* (or *clock variables*) to allow the modelling and analysis of a system behaviour related to timing constraints. Using timed automata, it is possible to state that a given automaton will leave state s_n before 10 time-units in the given state as shown in Figure 3.1 where c is a clock.

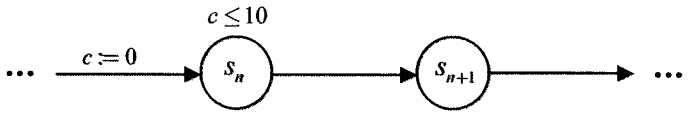


Figure 3.1: An Example of Timed Automata

With such controlling the behaviour of timing constraints in a system, timed automata have been extended and applied to real-time systems since their introduction.

There are many references which introduce timed automata in the literature [ABL98, AD90, AD94, BBF+01, BY04, CGP99, HNS+92] and use various terms and notations. So, for the convenience of the readers and to provide notations used for this work, the syntax and semantics of timed automata are defined below. There give exact meanings of terms such as clocks, clock constraint, guard, invariant, etc. Also, an example of timed automata model is presented to show how they may be used in a practical manner. The most important element in timed automata, the clock, is introduced first.

3.1.1 Clocks

Clocks in timed automata proceed at the same rate and are used to measure the progress of time. *A clock* is a variable ranging over non-negative real-value $\mathbb{R}$ and the set of such variables is denoted by *clocks* C. A clock works with two functions: *valuation* and *assignment. A valuation* is a function that assigns $\mathbb{R}$ to every clock. The set of valuations of C is denoted by $\mathcal{V}_C$ which means the set of all mappings from C to $\mathbb{R}$:

$$\mathcal{V}_C = \left[C \xrightarrow{all} \mathbb{R} \right]$$

The clock valuation is simply denoted by $v+d$, meaning that each clock $c \in C$ is increased by d with the valuation v where $v \in \mathcal{V}_C$ and $d \in \mathbb{R}$:

$$v+d = v(c)+d$$

An assignment is another function that assigns a clock to the value of another clock or 0. The set of assignments over C, denoted Γ_C, is the set of all mapping from C to C^0 where $C^0 = C \cup \{0\}$:

$$\Gamma_C = \left[C \xrightarrow{all} C^0 \right]$$

With $v \in \mathcal{V}_C$, $\gamma \in \Gamma_C$ and $c \in C$, $v(\gamma)$ denotes that each clock $c \in C$ is assigned to another clock:

$$v(\gamma)(c) = \begin{cases} v(\gamma(c)) & \text{if } \gamma(c) \in C \\ 0 & \text{otherwise} \end{cases}$$

3.1.2 Clock Constraints

The set of *clock constraints* over the set of clocks $\mathcal{C}$ is denoted by $\Psi_{\mathcal{C}}$ and the constraint $\psi \in \Psi_{\mathcal{C}}$ is defined by:

$$\psi := c \prec d \,|\, c - c' \prec d \,|\, \neg\psi \,|\, \psi \wedge \psi$$

where $c, c' \in \mathcal{C}, \prec \in \{<, >, =, \leq, \geq\}$ and $d \in \mathbb{R}$ is a natural number. Over clock valuations, a clock can be checked by, for example, $v(c) \prec d$. The notation $v \models \psi$ denotes the satisfaction of the clock constraint $\psi \in \Psi_{\mathcal{C}}$ over the clock valuation $v \in \mathcal{V}_{\mathcal{C}}$, the following satisfactions between clocks and clock constraints exist:

$$\begin{aligned}
v &\models c \prec d && \text{iff} && v(c) \prec d \\
v &\models c - c' \prec d && \text{iff} && v(c) - v(c') \prec d \\
v &\models \psi \wedge \psi' && \text{iff} && v \models \psi \text{ and } v \models \psi' \\
v &\models \neg\psi && \text{iff} && v \not\models \psi
\end{aligned}$$

There are two types of clock constraints. A clock constraint on transitions, called a *guard*, is used to determine the behaviour of transitions such as when a transition may proceed. A clock constraint on locations, called an *invariant*, means that a clock on a location can only stay as long as the invariant of the location is satisfied otherwise it has to leave from the location.

3.1.3 Timed Automaton

From these definitions of clocks and clock constraints, a timed automaton $\mathcal{A}$ can be defined as a tuple $(\mathcal{S}, \mathcal{C}, \mathcal{L}, \mathcal{I}, \mathcal{E})$ where:

- $\mathcal{S}$ is a finite set of locations. The initial location of $\mathcal{A}$ is distinguished from other locations as denoted by s_0.
- $\mathcal{C}$ is a finite set of clocks.
- $\mathcal{L}$ is a finite set of labels (or actions).
- $\mathcal{I} \in \{\mathcal{S} \rightarrow \Psi_{\mathcal{C}}\}$ is a finite set of mapping from locations to clock constraints. $\mathcal{I}(s)$ is referred as the invariant of $s \in \mathcal{S}$.

- $\mathcal{E}$ is a finite set of edges. Each edge $e \in \mathcal{E}$ is also defined as a tuple (s, l, ψ, γ, s') where:
  - $s \in \mathcal{S}$ is the start location.
  - $s' \in \mathcal{S}$ is the finish location.
  - $l \in \mathcal{L}$ is the label (or action).
  - $\psi \in \Psi_C$ is the enabling condition of a transition.
  - $\gamma \in \Gamma_C$ is the assignment.

3.1.4. Composition of Timed Automata

Most real-time systems can be modelled using several automata models rather than a single automaton; it is necessary to define the *composition of timed automata*. Let $\mathcal{A}_1 = (\mathcal{S}_1, \mathcal{C}_1, \mathcal{L}_1, \mathcal{I}_1, \mathcal{E}_1)$ and $\mathcal{A}_2 = (\mathcal{S}_2, \mathcal{C}_2, \mathcal{L}_2, \mathcal{I}_2, \mathcal{E}_2)$ be two timed automata models. When $\mathcal{C}_1 \cap \mathcal{C}_2 = \varnothing$ is assumed, the composition of two timed Automaton is defined by

$$\mathcal{A}_1 \| \mathcal{A}_2 = (\mathcal{S}_1 \times \mathcal{S}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{L}_1 \cup \mathcal{L}_2, \mathcal{I}(s_1, s_2), \mathcal{E})$$

where $\mathcal{I}(s_1, s_2) = \mathcal{I}_1(s_1) \wedge \mathcal{I}_2(s_2)$ as the invariant conjunction of the two timed automata models and $\mathcal{E}$ is further satisfied by the following conditions:

- With the same action of edges in $\mathcal{A}_1 \| \mathcal{A}_2$, there will be an edge which is a pair of edges and comes from both timed automata such that

  For $l \in \mathcal{L}_1 \cap \mathcal{L}_2$, (s, l, ψ, γ, s'), if $(s_1, l, \psi_1, \gamma_1, s_1') \in \mathcal{E}_1$ and $(s_2, l, \psi_2, \gamma_2, s_2') \in \mathcal{E}_2$, $\mathcal{E}$ will be $((s_1, s_2), l, \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, (s_1', s_2'))$
- With an action from one of timed automata in $\mathcal{A}_1 \| \mathcal{A}_2$, there will be a edge for each location of the other timed automaton such that

  For $l \in \mathcal{L}_1 - \mathcal{L}_2$, if $(s, l, \psi, \gamma, s') \in \mathcal{E}_1$ and $t \in \mathcal{S}_2$, $\mathcal{E}$ will has $((s,t), l, \psi, \gamma, (s', t))$

  For $l \in \mathcal{L}_2 - \mathcal{L}_1$, if $(s, l, \psi, \gamma, s') \in \mathcal{E}_2$ and $t \in \mathcal{S}_1$, $\mathcal{E}$ will has $((t,s), l, \psi, \gamma, (t, s'))$

3.1.5 State-Transition System

The semantics of a timed automata is given by associating a *state-transition system* $(\mathcal{Q}, \mathcal{L}, \rightarrow)$ where $\mathcal{Q}$ is a set of states and $\rightarrow \subseteq \mathcal{Q} \times \mathcal{L} \times \mathcal{Q}$ is a set of transitions, which is

the foundation for the verification of timed automata. Each state in Q is defined a pair (s,v) with a location and a valuation of clocks:

$$Q=\{(s,v)\in S\times V_c \mid v\models I(s)\}$$

Depending of the movement between states, there are two types of transitions in timed automaton. First, a transition corresponds to the elapse of time, called *delay transition.* It is defined by

$$(s,v)\xrightarrow{d}(s,v(c)+d) \text{ if } v\models I(s) \text{ and } v(c)+d\models I(s) \text{ where } d\in\mathbb{R}^+$$

Second, the other transition is called *action transition.* This transition corresponds to the execution of a transition, defined by

$$(s,v)\xrightarrow{l}(s',v(\gamma)) \text{ if } e=(s,l,\psi,\gamma,s')\in\mathcal{E},\ v\models I(s) \text{ and } v(\gamma)\models I(s')$$

Let a initial state of the timed automaton be (s_0,v_0), a run of a timed automaton intuitively appears as a sequence of states and transitions:

$$(s_0,v_0)\xrightarrow{l_0,d_0}(s_1,v_1)\xrightarrow{l_1,d_1}\ldots$$

3.1.6 An Example of a Timed Automata Model

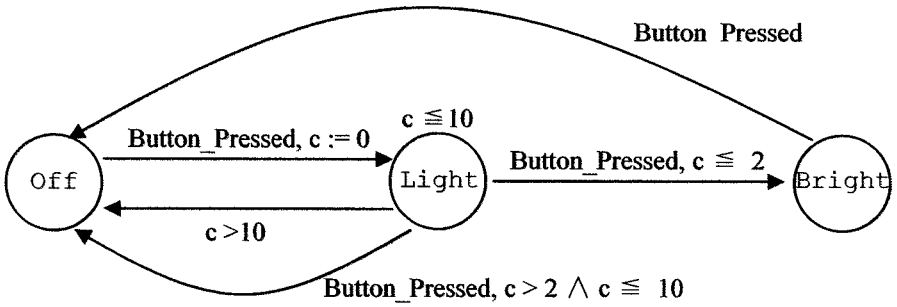


Figure 3.2: Timed Automaton for Intelligent Light Control

In order to give a better understanding of timed automata and their terms, a simple example of an intelligent light control system [LP00] is modelled by timed automata. The light simply behaves as follows. When the `Button_Pressed` is issued twice quickly, then the light will get brighter otherwise the light will be turned off some time later. After the light is bright, the additional `Button_Pressed` will make the light turn off again. Here, the most important thing is how to express the words related to time such as '*quickly*' and '*later*' in the system. Suppose that the words of '*quickly*' and '*later*'

mean that '*quickly*' is within 2 seconds and '*later*' is after 10 seconds. Concerning these timing constraints, the timed automaton for the light control system is shown in Figure 3.2.

The automaton consists of three locations labelled `Off`, `Light` and `Bright`, one clock, `c` and couple of transitions between the locations. In the `Off` location, the system waits until the event `Button_Pressed` occurs. As soon as the event has arrived, the automaton can make a transition from the `off` location to the `Bright` location and it resets the clock `c` to 0 during its transition. After then, the automaton has two choices depending on the arrival of the next `Button_Pressed`. First, when the next arrival is raised within 2 seconds, the automaton will move to the `Bright` location. Second, when the next arrival is greater than 2 seconds and less than or equal to 10 seconds, the automaton will move to the `Off` location. Otherwise the automaton in the `Bright` location will automatically go to the `Off` location and it starts again. In the `Bright` location the automaton can remain without timing constraint until `Button_Pressed` occurs.

3.2 Techniques for the Verification of Timed Automata

A timed automaton is a state-transition system with unbounded clocks. Such a system has possible infinite state spaces:

$$(s_0, v_0) \xrightarrow{l_0, d_0} (s_1, v_1) \xrightarrow{l_1, d_1} \ldots \xrightarrow{l_n, d_n} \ldots \infty$$

When such timed automata are evaluated, the infinite state space has to be constructed with bounded finite state regions.

3.2.1 Clock Regions

In order to convert infinite states to finite states in timed automata, an *equivalence relation* is introduced. The basic idea of an equivalence relation is that two states which are so related will produce a similar result in timed automata. Assuming that clock constraints work with only integers and also all clocks have the same rate of increase, it is defined that if two states in the same location of a timed automaton agree on the integral parts of all the clock values and the ordering of the fractional parts of the values, then two states will behave in a similar manner. The integral part is used to determine

whether or not a clock constraint in the invariant of a location or in the guard of a transition is satisfied, and the fractional part is to determine which clock will change its integral part first.

Let $cf(c)$ be a function, mapping each clock $c \in \mathcal{C}$ to the largest integer constant and let $\lfloor d \rfloor$ be the integral part of d, $fr(d)$ is the fractional part of d for a real number $d \in \mathbb{R}^+$ such as $d = \lfloor d \rfloor + fr(d)$. An equivalence relation $\sim$ is defined with two clock valuations v, v', denoted $v \sim v'$ if and only if the following three conditions are satisfied:

- For all $c \in \mathcal{C}$, either $\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor$ or $v(c) > cf(c)$ and $v'(c) > cf(c)$
- For all $c \in \mathcal{C}$ with $v(c) \leq cf(c)$, $fr(v(c)) = 0$ if and only if $fr(v'(c)) = 0$
- For all $c, c' \in \mathcal{C}$ with $v(c) \leq cf(c)$ and $v'(c') \leq cf(c')$, $fr(v(c)) \leq fr(v(c'))$ if and only if $fr(v'(c)) \leq fr(v'(c'))$

The equivalence class of an equivalence relation $\sim$ is called a *clock region* (or simply *region*) [AD90] which represents a set of clock assignments, denoted $[v]$. The equivalence classes are used as symbolic states to construct a finite-states transition system, called a *region graph* or a *region automaton* $\mathcal{R}e(\mathcal{A})$ of the original timed automaton; the transition on $\mathcal{R}e(\mathcal{A})$ is defined as follows:

- $(s,[v]) \Rightarrow (s,[v'])$ if $(s,v) \xrightarrow{d} (s,v')$ for a real number $d \in \mathbb{R}^+$
- $(s,[v]) \Rightarrow (s,[v'])$ if $(s,v) \xrightarrow{l} (s,v')$ for a label l under (s,l,ψ,γ,s')

As the transition on $\mathcal{R}e(\mathcal{A})$ is finite, the region automaton for a timed automaton has finite states.

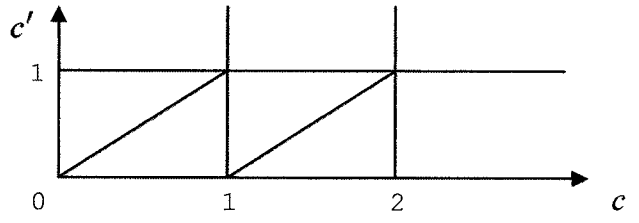


Figure 3.3: Regions with Two Clocks c and c'

Consider Figure 3.3 as an example which represents the set of regions with two clocks $c, c' \in \mathcal{C}$ and the largest number of c, $cf(c)$, is 2 and $cf(c')$ is 1 for c'. It is said that all corner points (e.g. $c = c' = 0$), line segments (e.g. $0 < c = c' < 1$), and open areas (e.g. $0 < c < c' < 1$) belong to regions. When counting the regions the Figure, the number of possible regions is expected to 28 regions, which comprise 6 corner points, 14 line segments and 8 open areas, i.e. a finite set of clock transitions.

3.2.2 Clock Zones

However, there are still problems in a region automaton $\mathcal{R}e(\mathcal{A})$. Depending on the number of clocks as well as the maximal constants in the guards of the automaton, $\mathcal{R}e(\mathcal{A})$ suffers from the potential explosion in the number of regions. A more efficient way to obtain finite states from the infinite states of a timed automaton is to use clock zones [BY04, CGP99] as a convex union of regions.

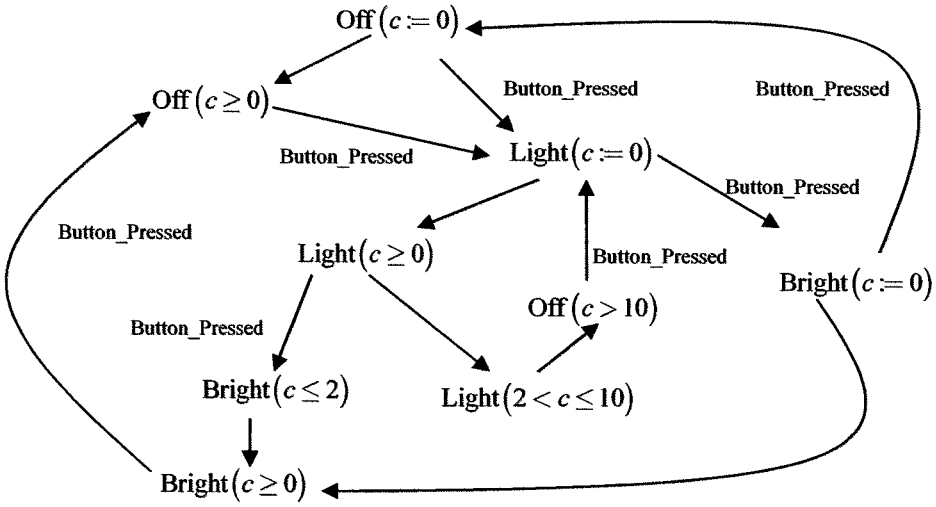


Figure 3.4: A Zone Automaton for Figure 3.2

A *clock zone* z is the maximal set of clock assignments satisfying a clock constraint, comparing either clock values or the difference between two clocks values to an integer. With a zone, it is possible to construct finite-states transition system called *a zone graph* or a *zone automaton*, $\mathcal{Z}o(\mathcal{A})$. A state of $\mathcal{Z}o(\mathcal{A})$ is a pair (s, z), where s is a location and z is a zone. The transition on $\mathcal{Z}o(\mathcal{A})$ is defined by the following rules:

- $(s,z) \Rightarrow (s,z')$ if $(s,v) \xrightarrow{d} (s,v')$ for a real number $d \in \mathbb{R}^+$ and $v, v' \in z$
- $(s,z) \Rightarrow (s,z)$ if $(s,v) \xrightarrow{l} (s,v')$ for a label l under (s,l,ψ,γ,s') and $v, v' \in z$

Consider the timed automaton shown in Figure 3.2 again as an example of a zone automaton. When it is presented to a zone automaton, it is noted that the automaton will have only 9 states and consequently has much less than a region automaton for the same timed automata. Figure 3.4 shows the zone automaton corresponding to the timed automaton in Figure 3.2.

3.3 The Application of Timed Automata to Schedulability

Due to their capability of verifying system behaviour, such as temporal constraints, timed automata have been used to model and verify the behaviour of scheduled systems. Several case studies found in the literature have already shown that timed automata can be used to verify systems using several automatic model checking tools such as UPPAAL [DBL+03, Hen02], KRONOS [BDM+98], etc. In particular, some case studies have tackled scheduling problems, one of the important issues in real-time system design.

Fehnker [Feh99] gave a model of a steel plant using UPPAAL based on a part of the SIDMAR steel production plant located at Gent in Belgium. It was a long term research project of the verification of hybrid systems to improve the take-up of modern information technologies in industry. It deals particularly with the part of the plant between the blast furnace and the continuous casting machine where molten pig iron is converted into steel of different qualities by various treatments. The model describes the behaviour of the plant based on the assumption that the transformation of pig iron to steel consists of a number of atomic, non-preemptive operations — a scheduling problem. In order to control the quality of the steel leaving the system within a given deadline, the model is used to determine whether a feasible schedule exist or not by verifying the model. In his work, he confirms that this approach, based on timed automata model, can use well known algorithms and a powerful formalism to model a scheduling problem via the UPPAAL verification tool; he emphasises that it is easily possible to add topological and timing constrains without being forced to change the underlying algorithms.

The same steel production plant was also modelled in the work of Hune *et al.* [HLP98]. With the model of the steel plant, they suggest a solution to the problem of scheduling and synthesising distributed control programs. In their work, the plant scheduling problem is formulated as a reachability question; the schedule is derived by again utilising the UPPAAL verification tool, which can provide a trace with actions of the model and timing information of actions after verifying the model by the reachability question. They synthesised executable control software using the trace. However, synthesising executable control programs requires the intense accuracy of the model such as requiring all necessary information of the timing bounds and the physical constraints for movements of loads, cranes, etc. The model easily becomes very complicated and thus the schedule for the model quickly becomes infeasible. For this reason, they introduce a method to overcome this infeasibility by guiding a model according to certain chosen strategies. Each strategy, such as some scheduling order is already given, contributes a reduction of the search-space. With this technique, they demonstrate that more batches can be synthesised in schedules.

A similar approach to the one presented above was used by Niebert *et al.* [NY00]. Using timed automata and model-checking algorithms, they suggest a similar method to automatically generate time-optimal production schemes particularly for a chemical batch plant. The plant considered in their work consists of containers, reactors, pipes, valves, pumps, etc., in terms for storing, transporting, processing and transforming raw materials to obtain a final chemical product; several products are concurrently manufactured just like operating in multi-batch mode. The chemical plant is modelled at the level of process operations whose behaviours are specified by timed automata extended with shared variables; the optimal production schemes are generated using algorithms for reachability analysis of timed automata implemented in the timing verification tool, KRONOS; the output of the verification tool, which is a sequence consisting of the elapsed time units and of transition of models, is post-processed to visualise the operation schemes and derive high-level control code. Still, the size of the state-space to be explored remains a serious obstacle; it is advised to utilise the knowledge of the plant to try to overcome it by guiding a particular resource when the resource has more possibility to produce successful schemes.

Altisen *et al.* [AGP+99] presented a framework integrating specification and scheduler generation for real-time systems. To describe the behaviour of real-time system, they use Petri Nets with Deadlines (PND) as modelling language in their framework which can facilitate the description of synchronisation conditions and enhance the readability of system specifications. Then, they choose Timed Automata with Deadlines (TAD) as semantic model of a PND. When given a TAD and a property for a system specification, synthesis algorithms allow computing the existence of a feasible schedule. They illustrate its applicability in practice with some example case studies: a greetings card, a system of three tasks and a robotic arm. However, they indicate that the framework approach is tractable only for medium size of systems because the approach has also a limitation resulting from state explosion.

TIMES [AFM+03] was recently developed at Uppsala University for schedulability analysis and to synthesise executable code based on the verification engine of the UPPAAL tool. This tool is particularly suitable for systems that can be described as a set of preemptive or non-preemptive tasks triggered periodically or sporadically by time or events. Each task is characterized by its worst execution time, deadline, priority, etc. and its execution may expect precedence and resource constraints. Timed automata are used to describe the schedulability of systems in the tool. When given a set of tasks characteristics and the timed automata model, TIMES will generate a scheduler and calculate the worst case response time for the tasks based on the selected scheduling policy. The various scheduling policies currently supported by the tool are: first-come first-served, rate monotonic, deadline monotonic and earliest-deadline first; all polices can be either preemptive or non-preemptive. The tool will further transform the automata model to executable code whose execution preserves the behaviour of the model, but unfortunately the code will work only on a limited range of platforms.

Many research have used timed automata to check the existence of certain paths among very large possible behaviours. Abdeddaïm *et al.* [AM01] are particularly interested in selecting a shortest path among them, assuming it as a time-optimal schedule. The classical job-shop scheduling problem, which has scheduling and resource allocation problems, is used to find a shortest one. In order to describe the way of establishing the link between the job-shop scheduling problem and time automata, they introduce many definitions such as job-shop specification, timed automata, feasible

schedule, etc. Interestingly, the definition of timed automaton for a job gives a simple idea of constructing timed automaton for many tasks together, an approach influencing the current study. They introduce and compare several algorithms in order to find the shortest path in such timed automata for job-shop scheduling; The symbolic forward reachability algorithm used in the tool KRONOS is adapted in their approach .

The state space explosion is a fundamental limitation of the timed automata approach when doing scheduling analysis, verification of real-time software, performance analysis, etc., and it certainly gives limitation of using timed automata. Thus, it is always desirable to avoid such a state explosion problem to improve the performance. Daws *et al.* [DY95] showed avoiding the difficulty of the verification of multirate timed automata by transforming multirate timed automata into normal timed automata using the timing verification tool KRONOS. Multirate timed automata are an extension of timed automata where each clock has its own speed varying between a lower and an upper bound, and thus are well-suited for specifying hybrid systems where the dynamics of the continuous variables are defined or can be approximated by giving the minimal and maximal rate of change. However, the automata have the difficulty in verification that the size of state-space for arbitrary time-bounds is exponential growth. This is the reason that they transform multirate timed automata to timed automata in order to handling the state-space problem. Two realistic case studies are used to show the practical interest of this approach: a manufacturing plant to find the set of all safe initial positions of the boxes and Philips Audio Control protocol to check all the invariants used in the protocol.

There are many efforts to ameliorate the state space explosion problem in the discrete verification literature even though it still remains an ongoing difficulty. Salah *et al.* [SBM06] proposed a solution to the state explosion problem caused by interleaving actions, merging the same set of interleaving actions and thus eliminating the interleaving explosion. With using local time scales, Bengtsson *et al.* [BJL+98] used distributed simulation which can compute successors for each automaton separately on its own clock, and then combine these local zones upon synchronisation. Daws *et al.* [DT98] have tried to reducing the number of zones using abstractions. There is still much discussion of concerning the improvement of state space explosion problem in verification but it is unfortunately beyond this research.

Much of the research aspects described above concerning schedulability using timed automata informs this current study. The approach proposed in this thesis has also employed the functionality of UPPAL to provide a trace after successfully verifying a model. Furthermore, the ways of constructing a timed automata model for tasks and for jobs here, which enables the consideration of scheduling problems for quite general systems.

3.4 Summary

This chapter has introduced the background of timed automata. First, the syntax and semantics of timed automata and their terms are formally defined to give their formal meanings; how to compose timed automatons and how to express them as state-transition system are discussed. The Intelligent Light Control is used to give an understanding of a timed automata model. Second, the infinite states problem in verification of a timed automaton has been described. As solutions for the problem, clock regions and clock zones are introduced. Finally, a number of studies which tackle scheduling problems using automatic model checking tools have been introduced. The knowledge introduced in this chapter will be important and fundamental to the approach, which will be proposed later, in order to automatically design schedules of distributed real-time systems in a time-triggered architecture.

Chapter 4

Scheduling with Timed Automata

In this chapter the scheduling of time-triggered architecture is described using timed automata. In particular, it focuses on the scheduling of distributed systems which involve many processors and possibly one or many networks. Typically such systems have many distributed tasks allocated to different processors which communicate via message passing on networks. First of all, the terms used throughout this thesis are formally defined to clarify the slightly different definitions found in the literature. And then, based on the formal definitions, schedules of the systems are constructed with timed automata.

4.1 Formal Definitions of the Terms

Since this research area is wide and varied, some terms are defined in slightly different ways. For example, in some cases, the terms of a process and task are synonymous, but in other cases, a process is more complex and referred to a set of many tasks. For this reason, it is worthwhile to define some terms first before further discussing the scheduling of systems. Consider the time-triggered scheduling in distributed real-time systems which may have many processors. Each *processor* connects to one or more *networks* and communicates by message passing on the networks. A set of *tasks* and *messages* are statically assigned to the processors and the networks respectively. The tasks and messages together can be further composed to *jobs* by specifying *precedence constraints*; these precedence constraints play a role of connection between tasks and messages. Feasible schedules can be formulated as a combinatorial problem of

satisfying the constraints of all jobs. Terms used throughout this thesis are formally defined as follows:

4.1.1 System

A *system* is defined by

$$S=(\mathcal{P},\mathcal{N},\mathcal{J})$$

where $\mathcal{P}$ is a finite set of *processors*, $\mathcal{N}$ is a finite set of *networks* and $\mathcal{J}$ is a finite set of *Jobs*. A processor $P_i \in \mathcal{P}$ is associated with a set of tasks, T_i, where $T_i \subseteq T$ and T is all tasks in S. The set of tasks on the processor P_i is denoted by T^{P_i} (or simply T_i). A network $N_i \in \mathcal{N}$ has a number of messages, M_i, where $M_i \subseteq M$ and M_i is all messages in S. M^{N_i} (or simply M_i) represents a set of messages associated with N_i. A job, $J_i \in \mathcal{J}$, is a sequence of tasks and messages imposed by precedence constraints; more discussion about a job will be placed in a later section. Figure 4.1 shows the graphical view of a system. The large rectangular regions represent processors and networks; the small circles and rectangles represent tasks and messages respectively which have to be executed or transmitted on a system.

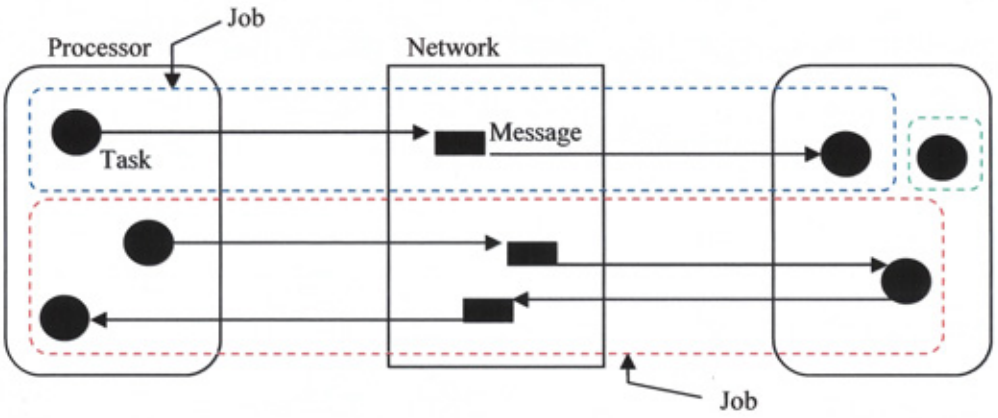


Figure 4.1: A Simple System expressed as a Precedence Graph

4.1.2 Task and Message

A *task* $t_i \in T$ is a tuple $t_i=(p,c)$ where p is the period of a task and c represents the worse case computation time. $t_i^{P_i}$ presents a task on the processor P_i. A *message*

$m_i \in M$ is tuple $m_i = (p, \pi)$ where p is the period of a message and π is the time for transferring a message. $m_i^{N_i}$ means a message on the network N_i. These tasks and messages have precedence constraints between them which are specified by their predecessors and successors. A task has only one predecessor or none if it is the initial task of a job, and also has one successor or none if it is the terminating task of a job; a message always has both of predecessor and successor.

4.1.3 Precedence Constraint

Let PC be a set of precedence constraints between tasks and messages, and $pc_i \in PC$ be the precedence constraint for t_i or m_i. The constraint pc_i specifies a predecessor and successor of a task t_i or message m_i. For this reason, a precedence constraint is represented as a pair, $pc = (\overrightarrow{pc}, \underrightarrow{pc})$, where $\overrightarrow{pc}$ indicates the preceding task or message, and $\underrightarrow{pc}$ indicates the succeeding task or message. Let ϕ be a null value which is used where a task does not have either a predecessor or successor. The following precedence constraints on a task t_i and message m_i are expressed in a system:

$$t_i \bullet \overrightarrow{pc} = \begin{cases} \phi & \phi \prec t_i \\ t_{i-1} & t_{i-1} \prec t_i \\ m_{i-1} & m_{i-1} \prec t_i \end{cases} \qquad t_i \bullet \underrightarrow{pc} = \begin{cases} \phi & t_i \prec \phi \\ t_{i+1} & t_i \prec t_{i+1} \\ m_{i+1} & t_i \prec m_{i+1} \end{cases}$$

$$m_i \bullet \overrightarrow{pc} = \{ t_{i-1} \quad t_{i-1} \prec m_i \qquad \mathrm{m}_i \bullet \underrightarrow{pc} = \{ t_{i+1} \quad m_i \prec t_{i+1}$$

For convenience, the relationship between tasks, messages and precedence constraints can be equivalently expressed as the tuples, $t_i = (p_i, c_i, pc_i)$ and $m_i = (p_i, \pi_i, pc_i)$, and the expressions $t_i \bullet pc$ and $m_i \bullet pc$ are used to describe their precedence constraints. For example, consider the following precedence constraints between tasks and messages.

$$\phi \longrightarrow t_0 \longrightarrow m_1 \longrightarrow t_2 \longrightarrow m_3 \longrightarrow t_4 \longrightarrow \phi$$

where $T = \{t_0, t_2, t_4\}$ and $M = \{m_1, m_3\}$. It is expected that the constraints are:

$$t_0 \bullet pc = (\phi, m_1) \qquad m_1 \bullet pc = (t_0, t_2) \qquad t_2 \bullet pc = (m_1, m_3)$$

$$m_3 \bullet pc = (t_2, t_4) \qquad t_4 \bullet pc = (m_3, \phi)$$

4.1.4 Job

The term "Job" is widely used mean a finite sequence of activities [But97, Red98]. Here it is specifically defined as a finite set of precedence constrained tasks and messages:

$$J_i = (p_i, T_i, M_i, PC_i)$$

where p_i is the period of J_i, and $T_i \subseteq T$, $M_i \subseteq M$ and $PC_i \subset PC$ are the sets of tasks, messages and precedence constraints respectively belonging to J_i. However, this definition needs to be cast into a form more suitable for expression as a timed automata model. So, let W be the set of $T \cup M$ for tasks and messages, and $w_i \in W$ is either a task or message. A job, $J_i \in \mathcal{J}$, may now be defined as a tuple:

$$J_i = (p_i, w_0, w_n)$$

where p_i denotes the period of a job, w_0 and w_n are the start and end of the sequence of tasks $(w_0, w_1, ..., w_n)$. It is assumed that the following conditions on a job have to be satisfied:

- The precedence constraints from w_0 to w_n on a job have to be correctly defined. Thus, $\forall i \ \ w_i \bullet \underrightarrow{pc} = w_{i+1} \bullet \overrightarrow{pc}$ for $0 \le i < n$ where $w_i \bullet pc$ is the precedence constraint of w_i.
- A job has to start with a task and end with a task ($w_0 \notin M$ and $w_n \notin M$).
- All tasks and messages belonging to a job have the same period: $\forall i, j \ \ w_i \bullet p = w_j \bullet p$ for $0 \le i < j \le n$ where $w_i \bullet p$ is the period of w_i.

4.1.5 An Example of Formal Expression

Consider the system, which comprises two processors, one network and three jobs, in Figure 4.1; with the above formal expressions, it is described as follows:

$$S = (\mathcal{P} = \{P_1, P_2\}, \mathcal{N} = \{N_1\}, \mathcal{J} = \{J_1, J_2, J_3\})$$

The processor P_1 and P_2 have three tasks each; the network N_1 also has three messages where:

$$P_1 = \left(T^{P_1} = \left\{t_1^{P_1}, t_2^{P_1}, t_3^{P_1}\right\}\right),\ P_2 = \left(T^{P_2} = \left\{t_1^{P_2}, t_2^{P_2}, t_3^{P_2}\right\}\right),\ N_1 = \left(M^{N_1} = \left\{m_1^{N_1}, m_2^{N_1}, m_3^{N_1}\right\}\right)$$

There are three jobs in the system where J_1 starts from $t_1^{P_1}$ and ends to $t_1^{P_2}$, J_2 is from $t_2^{P_1}$ to $t_3^{P_1}$ and J_3 is working itself. In accordance with the definition of a job, the three jobs can be expressed:

$$\mathcal{J} = \left\{ J_1 = \left(p_1, t_1^{P_1}, t_1^{P_2}\right), J_2 = \left(p_2, t_2^{P_1}, t_3^{P_1}\right), J_3 = \left(p_3, t_2^{P_2}, t_2^{P_2}\right) \right\}$$

The precedence constraints for jobs are implicitly defined based on their description. For example, $\overrightarrow{pc}$ of $t_1^{P_1}$ is a null value ϕ and $\underrightarrow{pc}$ of $t_1^{P_1}$ is $m_1^{N_1}$ as the succeeding step after $t_1^{P_1}$ completes. All the precedence constraints are:

$$t_1^{P_1} \bullet pc = \left(\phi, m_1^{N_1}\right),\ t_2^{P_1} \bullet pc = \left(\phi, m_2^{N_1}\right),\ t_3^{P_1} \bullet pc = \left(m_3^{N_1}, \phi\right)$$

$$m_1^{N_1} \bullet pc = \left(t_1^{P_1}, t_1^{P_2}\right),\ m_2^{N_1} \bullet pc = \left(t_2^{P_1}, t_3^{P_2}\right),\ m_3^{N_1} \bullet pc = \left(t_3^{P_2}, t_3^{P_1}\right)$$

$$t_1^{P_2} \bullet pc = \left(m_1^{N_1}, \phi\right),\ t_2^{P_2} \bullet pc = \left(\phi, \phi\right), t_3^{P_2} \bullet pc = \left(m_2^{N_1}, m_3^{N_1}\right)$$

Figure 4.2 shows the system with all the formal expressions.

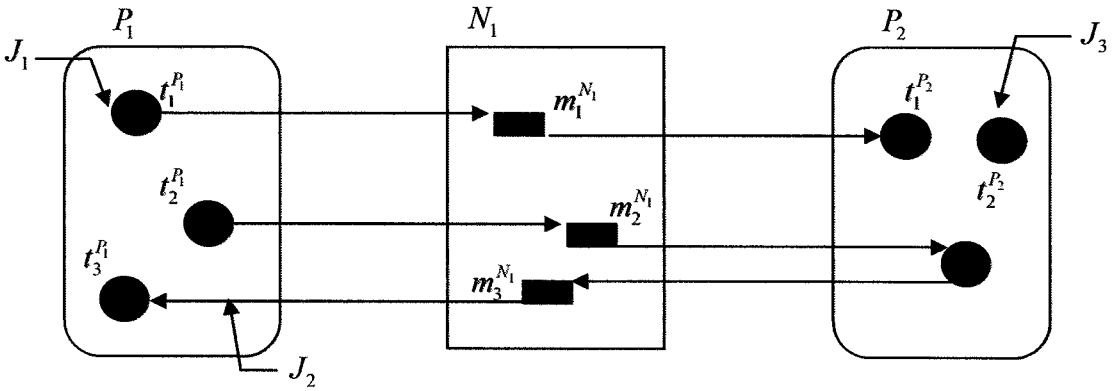


Figure 4.2: The Formal Expression of the Simple System

4.2 Design Principle of Schedules with Timed Automata

Based on the basic design principle that a task and message can be modelled with finite states and a clock variable, a job consisting of tasks and messages, formally defined by $J_i = \left(p_i, w_0, w_n\right)$, can be constructed as a timed automata model with finite states and clocks. Even by composing the models for jobs, it is further possible to construct the behaviour of a system in timed automata and to recognise a system schedule.

4.2.1 A Timed Automata Model for a Task and Message

For any task and message in J, denoted $J(w)$ and simply called a *step*, it is proposed that each step can be transformed into three essential states in timed automata. $J(w)$ is defined $J(w)=(q_{\bar{w}}, q_w, q_{\underline{w}})$ where $q_{\bar{w}}$ is the initial state, q_w is the active state, and $q_{\underline{w}}$ is the final state. These states denote that $q_{\bar{w}}$ is a state waiting for the start of w; q_w is a state representing the current work of w; $q_{\underline{w}}$ is a state waiting for the succeeding step after w is completed. In timed automata, the three states are depicted with transitions in Figure 4.3.

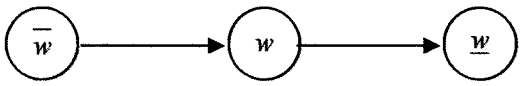


Figure 4.3: Three States of each Step

Based on the three states, a clock is added on the transitions being the condition for them to make a shift from one state to another. Such as, when $q_{\bar{w}}$ state enters to q_w, the clock is reset to time 0 in order to measure the active time of a current step. q_w can move to $q_{\underline{w}}$ only if the clock satisfies the condition that the computation time of a task or the transfer time of a message is equal to the clock value. With the function $\tau(w)$ denoted the required execution time of w, the completion of the timed automata model including the states and transitions is shown in Figure 4.4.

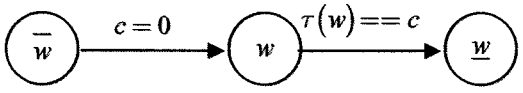


Figure 4.4: The States for Each Step with Clock

4.2.2 A Timed Automata Model for a Job

A job is a finite sequence of steps and is formally defined as a tuple $J_i=(p_i, w_0, w_n)$, particularly describing the initial step and final step of J. This means that starting from the initial w_0, it is possible to trace the succeeding step of w_0 and even the successive

step of the successive step of w_0; eventually the trace will reach the final w_n of J. It is an assumption that a precedence constraint in each step is correct. With the automata model for a task and message, a job also is expressed as a sequence of states:

$$J=\left(d,w_0,w_f\right) \Rightarrow w_0 \rightarrow \ldots \rightarrow w_i \rightarrow \ldots \rightarrow w_n$$

$$\Rightarrow \left(q_{\overline{w_0}},q_{w_0},q_{\underline{w_0}}\right)\rightarrow \ldots \rightarrow \left(q_{\overline{w_i}},q_{w_i},q_{\underline{w_i}}\right)\rightarrow \ldots \rightarrow \left(q_{\overline{w_n}},q_{w_n},q_{\underline{w_n}}\right)$$

Consequently, the timed automata model of a job transformed with states is shown in Figure 4.5.

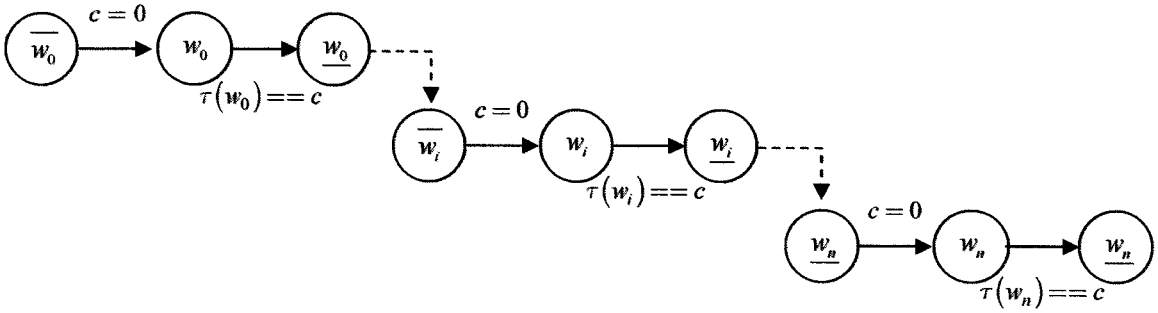


Figure 4.5: The Automata Model for a Job

However, supposing $\underline{w_i}=\overline{w_{i+1}}$ meaning that the finial state of w_i can be treated as the initial state of w_{i+1}, it is possible to reduces many states in the model and express it more simply. As the result, it is required to change the transition conditions in the model as follows:

$$\begin{cases}\overline{w_i}\rightarrow w_i & \Rightarrow \quad c=0\\ w_i\rightarrow \overline{w_{i+1}} & \Rightarrow \quad r\left(w_i\right)==0\end{cases}$$

Figure 4.6 shows the automata model which is reduced by many states, so: $\underline{w_i}=\overline{w_{i+1}}$.

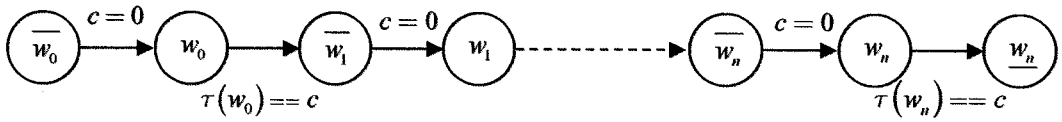


Figure 4.6: The Reduced Automata Model for a Job

There is further improvement of the model in a case that a job repeats its arrival every period. The efficient way for this improvement is to add a transition between the end state $\underline{w_n}$ and the start state $\overline{w_0}$, moving $\underline{w_n}$ to $\overline{w_0}$ and then to set a condition only allowing the movement of the transition when a clock value reaches to the period of a job. In other words, it forces the model waiting for the next arrival. See the Figure 4.7 showing the repetition over the model.

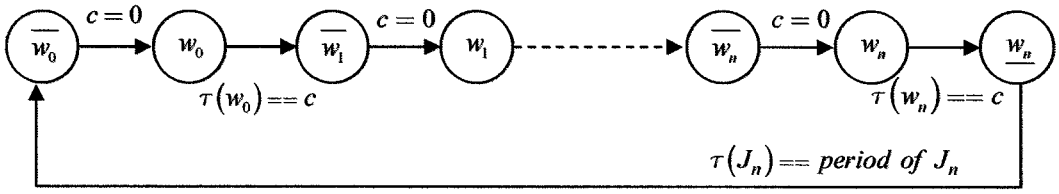


Figure 4.7: The Automata model for a Job with Repetition

4.2.3 A Timed Automata Model for a System Schedule

With the timed automata models for all jobs in a system, it is possible to construct a system schedule in timed automata. It is simply to compose the models for all jobs and to work together. However, it is intuitively expected that the composing model may be huge and complex despite composing just a few jobs. This is because in the composition, each state has not one transition but has many possible transitions to progress. In addition, the conflicting states have to be avoided as a processor and network do not allow multi-access at the same time, required by several jobs.

Figure 4.8 shows a possible automata model corresponding to a system schedule. It starts with $\left(\overline{w_0^1}, \overline{w_0^2}, \ldots, \overline{w_0^n}\right)$ state representing the initial state in which all jobs are ready to work such that $\overline{w_0^1}$ is the initial state of J_1, $\overline{w_0^2}$ is the initial state of J_2 and so on. Then this state is possible to move to one of active states among such states $\left\{\left(w_0^1, \overline{w_0^2}, \ldots, \overline{w_0^n}\right), \left(\overline{w_0^1}, w_0^2, \ldots, \overline{w_0^n}\right), \ldots, \left(\overline{w_0^1}, \overline{w_0^2}, \ldots, w_0^n\right)\right\}$ but the conflicting states, of course, have not to be considerable as active states. For example, supposing that the current state is $\left(w_0^1, \overline{w_0^2}, \ldots, \overline{w_0^n}\right)$ and $\overline{w_0^2}, \ldots, \overline{w_0^n}$ are conflicting states, it is only possible

to move to $\left(\overline{w_1^1}, \overline{w_0^2}, \ldots, \overline{w_0^n}\right)$ unlike the many transitions shown in Figure 4.8. This is because other jobs also require the resource taken by w_0^1, so they have to wait for it until it is released by w_0^1. Avoiding conflicting states, it is possible to construct the timed automata model of a system schedule. However, creating the model by hand, even with just a few jobs, requires considerable effort. For this reason, creating such a model for a system schedule is not recommended but is advised to use model-checking tool. There will be further discussion of using the tool later.

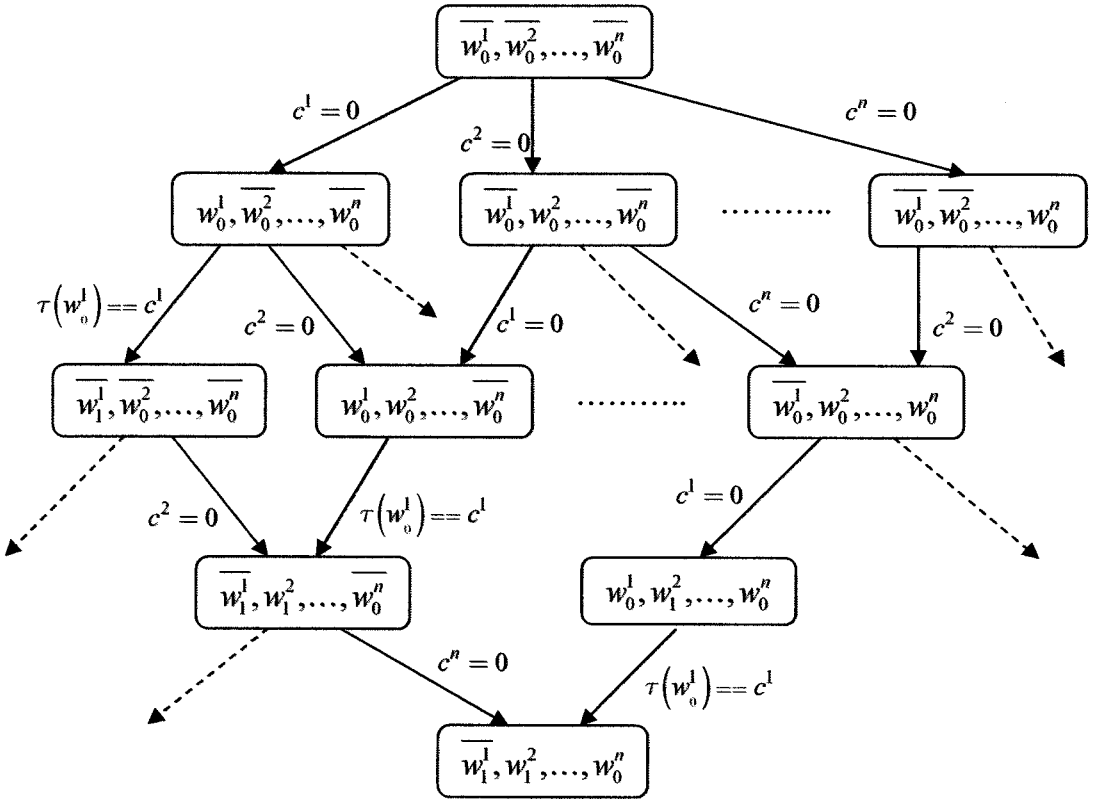


Figure 4.8: The Automata Model for a System Schedule

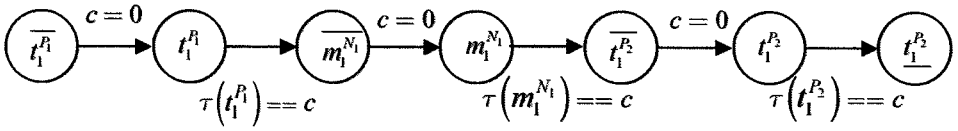
4.2.4 An Example of a Timed Automata Model for a Job

Consider the system $S = \left(\{P_1, P_2\}, \{N_1\}, \{J_1, J_2, J_3\}\right)$ again shown above to express a job in a timed automata model. Recalling the system briefly, it has two processors, a network and three jobs; the processor P_1 and P_2 have the set of tasks, $T^{P_1} = \left\{t_1^{P_1}, t_2^{P_1}, t_3^{P_1}\right\}$

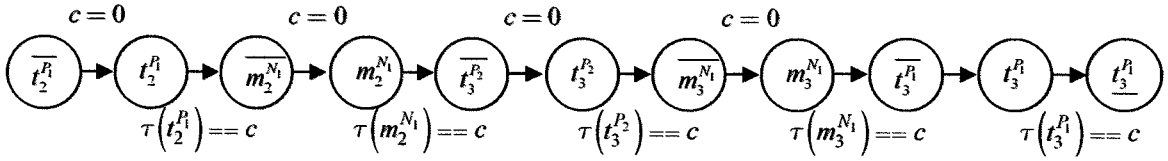
and $T^{P_2} = \{t_1^{P_2}, t_2^{P_2}, t_3^{P_2}\}$ respectively; the messages on the network N_1 are $M^{N_1} = \{m_1^{N_1}, m_2^{N_1}, m_3^{N_1}\}$; there are three jobs in the system: $J_1 = (p_1, t_1^{P_1}, t_1^{P_2})$, $J_2 = (p_2, t_2^{P_1}, t_3^{P_1})$ and $J_3 = (p_3, t_2^{P_2}, t_2^{P_2})$.

Since a job is defined with a start and end task to express it as a finite sequence of steps as known tasks and messages, it is expected that J_1 and J_2 have the following sequences: $t_1^{P_1} \rightarrow m_1^{N_1} \rightarrow t_1^{P_2}$ and $t_2^{P_1} \rightarrow m_2^{N_1} \rightarrow t_3^{P_2} \rightarrow m_3^{N_1} \rightarrow t_3^{P_1}$ respectively, but J_3 has a single task $t_2^{P_2}$ working alone. On the basis of the sequence of steps for each job, each step of the sequence can be transformed into three essential states and eventually the sequence known as a job will be created to a timed automata model having a sequence of states. These three jobs are expressed in timed automata in Figure 4.9.

$J_1 = (p_1, t_1^{P_1}, t_1^{P_2}) \Rightarrow t_1^{P_1} \rightarrow m_1^{N_1} \rightarrow t_1^{P_2}$



$J_2 = (p_2, t_2^{P_1}, t_3^{P_1}) \Rightarrow t_2^{P_1} \rightarrow m_2^{N_1} \rightarrow t_3^{P_2} \rightarrow m_3^{N_1} \rightarrow t_3^{P_1}$



$J_3 = (p_3, t_2^{P_2}, t_2^{P_2}) \Rightarrow t_2^{P_2}$

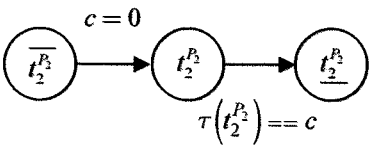


Figure 4.9: The Timed Automata Models for the Jobs in the System.

Although the timed automata model for the schedule of this system will not be created but rather be executed using model-checking tool, the simplified model for that may be expected, shown in Figure 4.10. The model considers the three jobs collectively

and starts with $\left(\overline{t_1^{P_1}}, \overline{t_2^{P_1}}, \overline{t_2^{P_2}}\right)$ state representing the initial state for all the three jobs. The initial state can move any following state choices $\left\{\left(t_1^{P_1}, \overline{t_2^{P_1}}, \overline{t_2^{P_2}}\right), \left(\overline{t_1^{P_1}}, t_2^{P_1}, \overline{t_2^{P_2}}\right), \left(\overline{t_1^{P_1}}, \overline{t_2^{P_1}}, t_2^{P_2}\right)\right\}$; if the initial state moves to $\left(t_1^{P_1}, \overline{t_2^{P_1}}, \overline{t_2^{P_2}}\right)$ state, this means that J_1 starts first; moving to $\left(\overline{t_1^{P_1}}, t_2^{P_1}, \overline{t_2^{P_2}}\right)$ state means that J_2 starts first; $\left(\overline{t_1^{P_1}}, t_2^{P_1}, t_2^{P_2}\right)$ state is the start of J_3. In spite of considering the three jobs, the model will have a lot of choices to find the schedule of the system. For example, when considering the execution of J_1 first on the model, the choices may follow the red arrows in Figure 4.10.

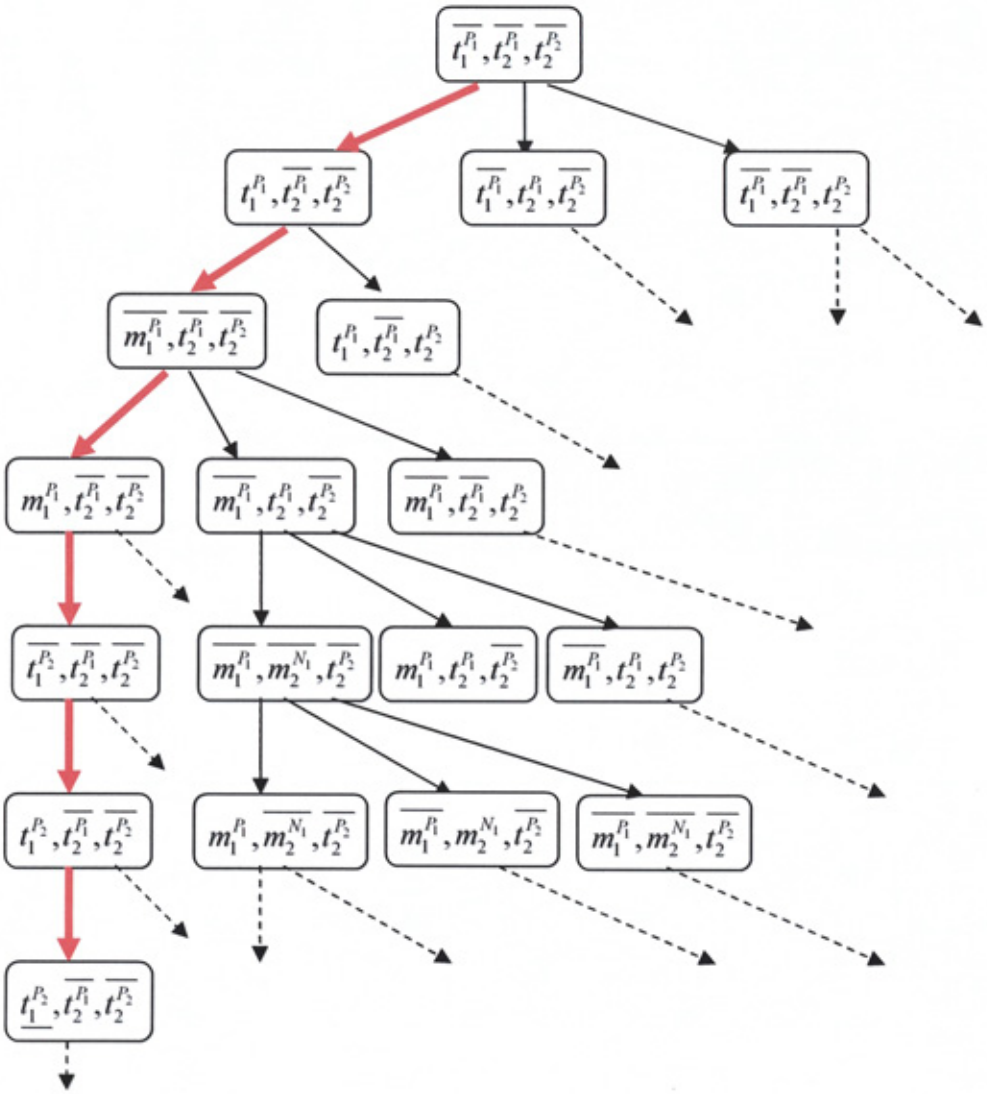


Figure 4.10: The Execution of J_1 first in the Model for the System Schedule

4.3 Feasible Schedules from Timed Automata

4.3.1 Feasible Schedules

In order to generate schedules in a time-triggered architecture, it is essential to find an adequate scheduling period for all jobs, called a *scheduling round* $\mathbb{R}$ or simply a r*ound*. The round is the time window within which all jobs have to finish; it repeats over and over again in a system. Finding the round with the different periods of jobs, however, has to be considered because the round is required to include the periods of all jobs. It is generally known that the simplest way to find the round is the *Least Common Multiple* of the periods. In order to find such a time window using *LCM* easily it is recommended that the periods of all jobs should be integer numbers rather than real numbers. And so, this enables the calculation of the time window in which all jobs are to be executed an integral number of times [Red98].

Given an acceptable round for a system, it is possible to define a feasible schedule S_{ch} for all jobs as a relation:

$$S_{ch} \subseteq \mathcal{J} \times \mathbb{C}$$

where $\mathcal{J}$ is a set of jobs on a system and $\mathbb{C}$ is a set of time values satisfying the timing requirement of $\mathcal{J}$. $(J,c) \in S_{ch}$ indicates that the job J is allocated with the time c, meaning that each task and message belonging to the job have the correct starting time for their execution. However, there is a caution in a feasible schedule such that a job must consider *mutual exclusions* because some jobs may require the resource of a processor simultaneously. This means that when $J_i \bullet w_n$ and $J_j \bullet w_n$ require the same resource at the same time, the time allocated to a job in a feasible schedule, S_{ch}, should satisfy the following condition:

$$finish\left(\mathrm{Res}_i\left(J_i \bullet w_n\right)\right) < start\left(\mathrm{Res}_i\left(J_j \bullet w_n\right)\right) \text{ and vice versa for } J_i, J_j \in \mathcal{J}$$

where $\mathrm{Res}_i\left(J_i \bullet w_n\right)$ be a function representing the use of Res_i resource requested from $J \bullet w_n$, $start(J_i \bullet w_n)$ a function indicating the start time of $J_i \bullet w_n$ and $finish(J_i \bullet w_n)$ a function indicating the start time of $J_i \bullet w_n$.

4.3.2 Finding Feasible Schedules

Working with the timed automata models representing a system, it is likely that many feasible schedules S_{ch} will be derived and enable all jobs to finish within a scheduling round. It might be one or a couple of or a lot of schedules depending on the size of the given round. Consider the system shown in the Figure 4.1 and assume that all the jobs in the system are given 90 time-units as their periods – intuitively the round is the same as the periods. In this case, many feasible schedules are likely to be expected. Using a Gantt diagram, there are two schedules considering the system shown in Figure 4.11. The two schedules S_{ch1} and S_{ch2} both satisfy all the jobs and complete within 90 time-units but S_{ch1} can finish all the jobs 10 time-units earlier than S_{ch2} schedule. It is because S_{ch1} executes some tasks and messages simultaneously but S_{ch2} has not.

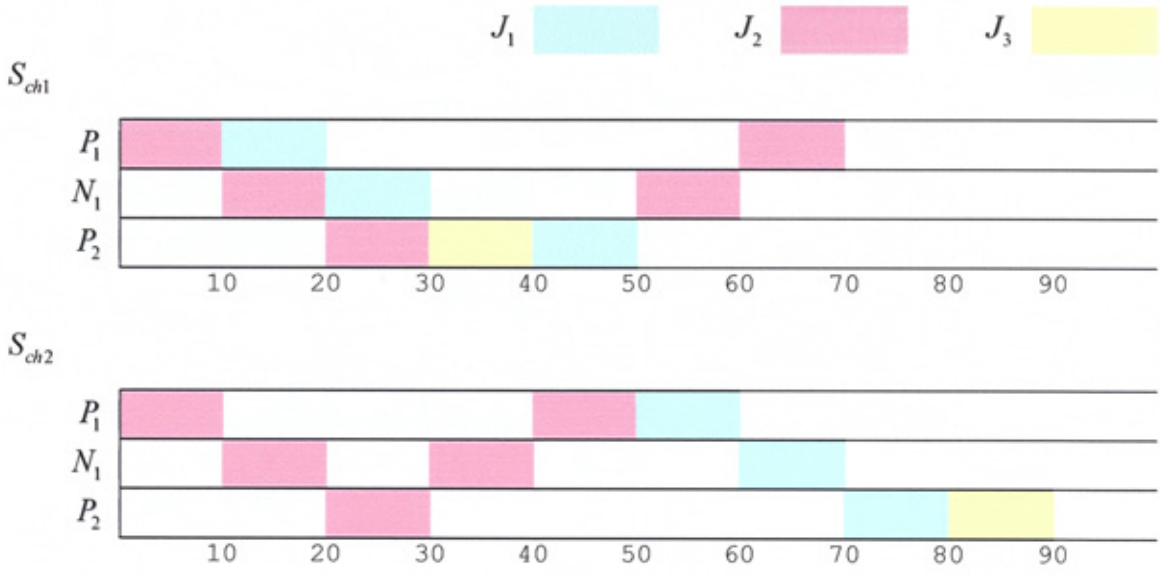


Figure 4.11: The Schedules of S_{ch1} and S_{ch2}

These two schedules are certainly accepted as feasible schedules since they produce a set of start times for the jobs within the given round, satisfying the jobs with such following times: in S_{ch1}, J_1 starts at 10 time-units and finishes at 50 time-units; J_2 works from 0 to 70 time-units; J_3 is between 30 and 40 time-units; in S_{ch2}, J_1 is from 50 time-units to 80 time-units; J_2 works between 0 to 50 time-units; J_3 is between 80

to 90 time-units. It is further expected that the tasks and messages belonging to the jobs have automatically the following start and end times:

J_1	S_{ch1}	S_{ch2}	J_2	S_{ch1}	S_{ch2}	J_3	S_{ch1}	S_{ch2}
$t_1^{P_1}$	10	50	$t_2^{P_1}$	0	0	$t_2^{P_2}$	30	80
$m_1^{N_1}$	20	60	$m_2^{N_1}$	10	10			
$t_1^{P_2}$	40	70	$t_3^{P_2}$	20	20			
			$m_3^{N_1}$	50	30			
			$t_3^{P_1}$	60	40			

Table 4.1: The Schedule for Tasks and Messages in S_{ch1} and S_{ch2}

Among the derived feasible schedules for the system, it may be considered that the schedules which have the shortest length are desired; this makes a system compact and tight. These shortest schedules may help the extensibility and flexibility for systems in cases of adding or modifying tasks on a schedule and they are denoted by $S_{sht} = \min_{i \in n} finish(S_i)$ where $finish(S_i)$ is a function indicating the finish time of S_i. However, these schedules are not easy to find out straightforwardly because many factors are required to consider such as the order of tasks and messages, the resource arrangement, etc.

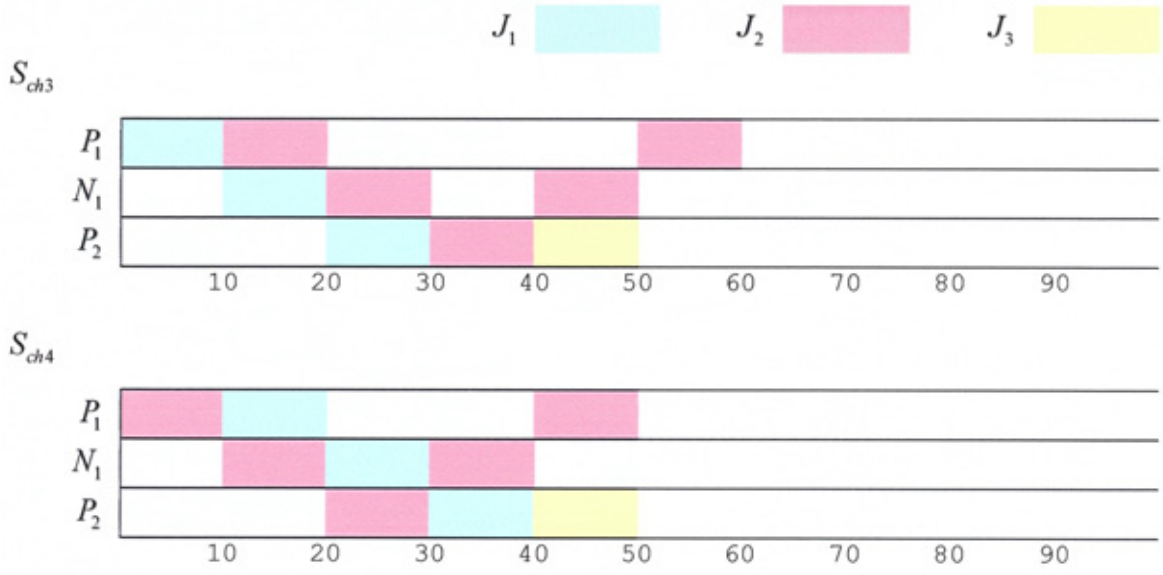


Figure 4.12: The Schedules of with S_{ch3} and S_{ch4}

When considering the system in the Figure 4.1 again, there are further two schedules, S_{ch3} and S_{ch4}, constructed, in particular, both allowing the resource access without any delay when it is requested. As depicted in Figure 4.12, the difference between two schedules is that S_{ch3} makes J_1 start first and then followed by J_2, but S_{ch4} does the reverse. Although the difference is only the execution order of the jobs, it has affected the two schedules. According to the Figure, S_{ch3} has finished all the jobs within 60 time-units and S_{ch4} is 50 time units. This is understood that S_{ch4} is more compact and tighter than S_{ch3}, only affected by the execution order. Such simple systems like the system in Figure 4.1 will be easy to find S_{sht} and even possible to find it by inspection. But in complicated systems having many tasks and messages, finding S_{sht} is almost impossible manually and often required great effort such as using automatic algorithms, methods and tools.

4.4 Summary

In order to ameliorate the confusion of the terms differently referred in the literature, this chapter has established formal definitions of terms used to construct timed automata models. With the principle that a task and message can be transformed into three essential states and a clock, a job consisting of tasks and messages has been formalised in timed automata. Further, this chapter has introduced how to model the behaviour of a system in timed automata, composing the timed automata models for jobs. Given a scheduling round for systems in a time-triggered architecture, feasible schedules have been discussed with examples. In particular, the schedules which have the shortest length have been considered to give greater extensibility and flexibility to systems.

Chapter 5

Efficient Timed Automata Models for Scheduling

In the previous chapter, the job comprising a sequence of tasks and messages has been formalized by timed automata based on the principle that a task or message can be modelled by finite states and a clock variable; the behaviour of an entire distributed system, thus, has been presented a parallel composition of these timed automata models. This chapter has further exploration to creating a real model using UPPAAL tool which is one of timed automata tools for symbolic model checking of real-time system. In order to ameliorate the main obstacle encountered during the model checking, i.e. state explosion problem, three slightly different models have been suggested and examined regarding their efficiency for the problem.

5.1 Overview of UPPAAL

5.1.1 UPPAAL

UPPAAL is a timed automata tool for symbolic model checking of real-time systems developed jointly by Uppsala University and Aalborg University. It supports formal verification and simulation of the behaviour of systems by checking invariant and reachability properties, and even provides facilities to detect deadlocks [Hun01, LPY98]. It is suitable for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables [LPY97]. Based on the notion of timed automata introduced by Alur et al. [AD90, AD94], UPPAAL provides an easy way of modelling systems, extending timed automata with more general types of data variables such as integer, boolean like those provided by high-level real-time languages.

UPPAAL has been applied in several industrial case studies such as real-time protocols, multi-media synchronization protocols, real-time controllers and proving the correctness of electric power plants and so on. Havelund et al. [HSL+97] apply UPPAAL in the context of audio and video protocols and prove the correctness of the protocols. UPPAAL is used to formally verify a time division multiple access (TDMA) based protocol intended for local area networks that operates in modern vehicles by Lönn et al. [LP97]. It also helps to support the development of system in automotive industry such as a Gear Controller [LPY98].

The UPPAAL tool kit includes three main parts: description language, simulator and model checker. The description language is used to model system behaviour with clocks and data variables. The simulator and model checker are used together for manipulating and solving constraints representing the state-space of a system description. They are able to check the behaviour of system and to produce valid counter examples for proof if requested; the temporal logic properties are used for model checking.

5.1.2 Syntax

A UPPAAL model is based on the notion of timed automata introduced in [AD94] and provides a more expressive model by introducing extensions such as *channel*, *urgent channel*, *committed location*, etc. Here, the syntax of UPPAAL [BLL+96, LPY97] is provided in more detail.

Guards. Guards express conditions on the values of clocks and integer variables on transitions that must be satisfied in order to take the transition. A guard may have two types of constraints: *timing constraints* and *data constraints*; a timing constraint is formally $c \prec n$ or $c - c' \prec n$ where $c, c' \in \mathcal{C}$ are clocks, $\prec \in \{<, >, =, \leq, \geq\}$ and $n \in \mathbb{N}$ is a natural number; a data constraint is almost the same as a timing constraint but has arbitrary integer, forming $a \prec i$ or $a - a' \prec i$ where $a, a' \in \mathcal{D}$ are a data variable, $\prec \in \{<, >, =, \leq, \geq\}$ and $i \in \mathbb{I}$ is an integer number.

Synchronisation. An UPPAAL model is a network of timed automata which can communicate using a binary synchronisation function, called a *channel* based on the classical message passing model. Given a communication channel a, there are two notations, $a!$ and $a?$, to perform sending a synchronisation to the channel a and receiving it from the channel a respectively. It is further possible to declare an urgent channel in order to prevent any delay between the synchronisation and thus time does not pass during its operation, i.e. time on clocks will not be changed.

Assignments (or Updates). Clocks and data variables are reset or assigned to a value on edges between locations. In UPPAAL, a clock variable must be assigned with integer expression using an assignment operator $:=$ but a data variable is allowed various expressions, the same as high-level languages.

Committed Locations. UPPAAL has the notion of committed locations in timed automata. When a location is declared as a committed location, it must immediately take on the next outward transition with no delay, the same as an urgent channel.

Invariants. As a guard on transitions, a location may possess a guard to control the automata model. This is called an *invariant*. To remain in a location, the invariant must remain true. The syntax for invariants and guards is the same.

5.1.3 An Example of a UPPAAL model: Olympic Boxing Scoring System

5.1.3.1 Description

There is an example of UPPAAL model, Olympic Boxing Scoring system. As known Olympic boxing (amateur boxing) generally, a boxing match consists of three 3-minute rounds and 1-minute interval between the rounds. Since 1992, the Olympic Committee have decided to use an electronic scoring system rather than scoring by hand. Under the electronic scoring system [IBA03], five judges hold a keypad which has red and blue scoring button. During a boxing match, judges record points for each competitor's blow using their keypad. Basically a point is recorded when three of five judges must press the same colour button within one-second interval. The interval begins when any judge presses a button first for recording a point. The competitor, who has most scores, has a win over the three rounds.

5.1.3.2 Modelling in UPPAAL

The Olympic Boxing Scoring system has three sub-models called *templates* in UPPAAL: *Boxing Scheduler*, *Judge* and *Score Controller* shown in Figure 5.1, 5.2 and 5.3 respectively.

Boxing Scheduler has four locations: `Start`, `OneRound`, `BreakTime` and `EndMatch`. The initial location is named as `Start` and committed location which means that once it starts, it moves to immediately `OneRound` location with informing a round start (`startRound!`) to all judges (`startRound?`) by using a broadcast channel and also resets the clock (`rndClock`) to zero. A round variable is used to count the number of rounds. After the clock (`rndClock`) passes one round time, it moves to the `BreakTime` or `EndMatch` location depending on the value of the round variable. If the `round < RND`, valued 3, it will move to the `BreakTime` location otherwise the match will be end by moving to the `EndMatch` location.

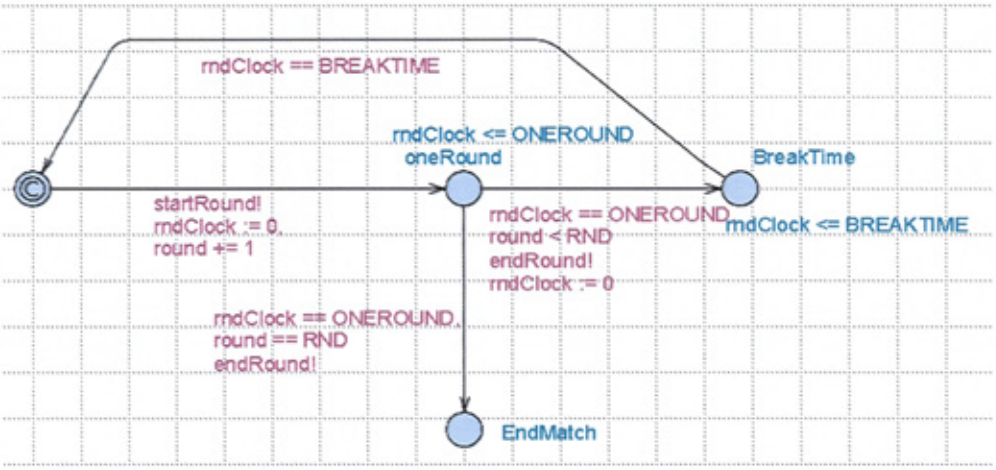


Figure 5.1: Boxing Scheduler Model

Judge is simple. After a start round (`startRound?`) is informed by Boxing Scheduler, each judge has two choices: pressing a red button or a blue button. When a judge presses a red button (`pressRed!`) or blue button (`pressBlue!`), it will be sent to Score Scheduler and each judge's ID is saved in the `whoPressed` variable. A judge will become inactive when an end round (`endRound?`) is sent by the Boxing Scheduler.

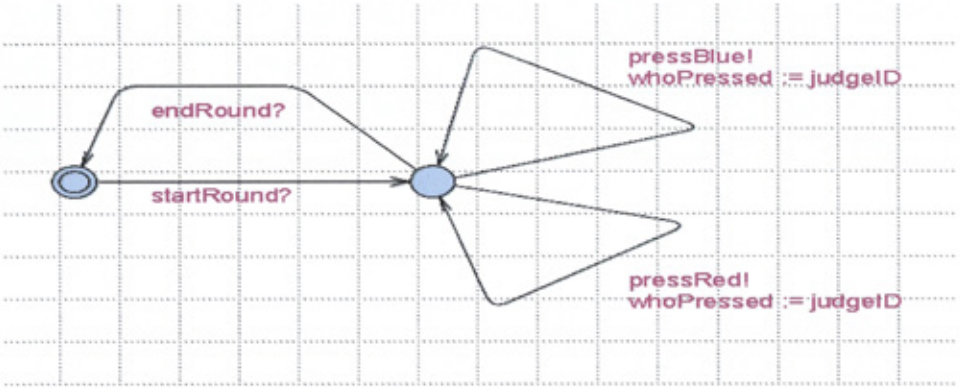


Figure 5.2: Judge Model

Score Controller has four locations: `WaitingFirstJudge`, `ScoringPeriod`, `Scoring` and `ClearJudgeScore`. Once one of judges presses a button, the initial location, `WaitingFirstJudge`, moves to `ScoringPeriod` and then keeps recording judges' score until INTERVAL, valued 1, has gone. The Scoring location has the role of counting the total number of how many judges have pressed during the INTERVAL. If `totalJudge > 3`, a point will be awarded otherwise it is ignored. For next scoring a point, the current scores for all judges during the interval are automatically cleared.

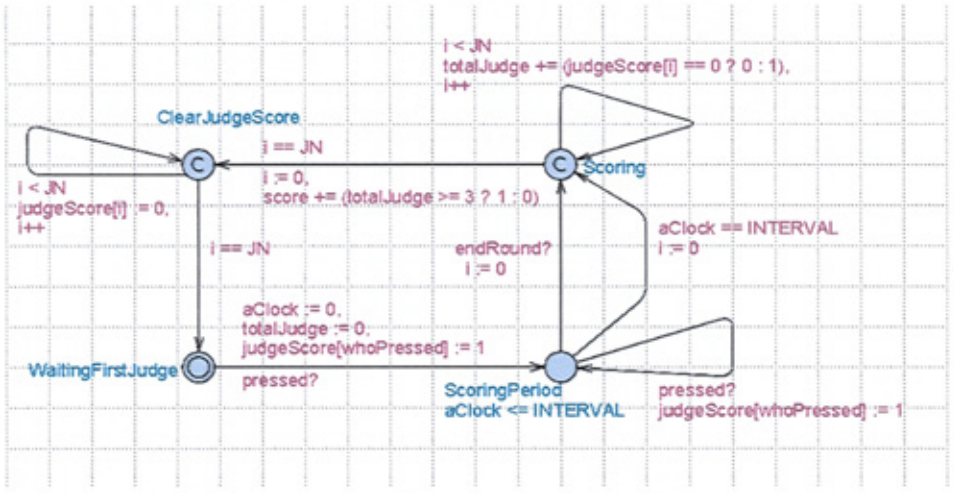


Figure 5.3: Score Controller Model

Based on the models created above, it is possible to define a scoring system corresponding to five Judges, two Score Controllers and one Boxing Scheduler in UPPAAL.

Process assignments

```
Judge0 := Judge(0);
Judge1 := Judge(1);
Judge2 := Judge(2);
Judge3 := Judge(3);
Judge4 := Judge(4);

redScoreBoard := ScoreController(pressRed, redScore);
blueScoreBoard := ScoreController(pressBlue, blueScore);

Scheduler := BoxScheduler();
```

Figure 5.4: Process assignments for the System

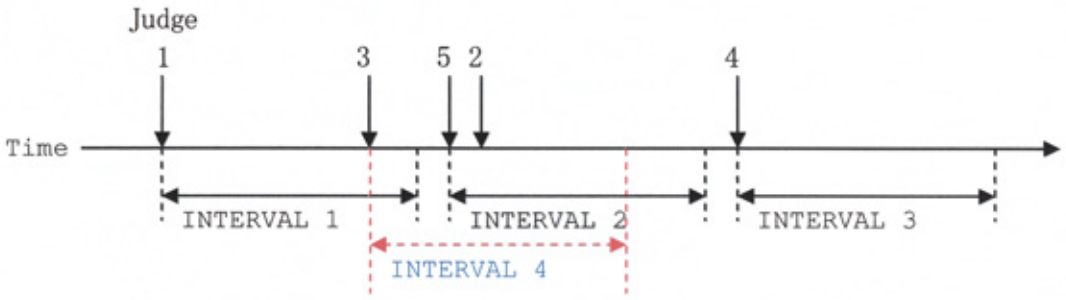


Figure 5.5: Score Controller Model

An interesting problem in the Score Controller model has been identified [Bur07]. As explained previously, the Score Controller model starts recording judge's score when one of the judges press a button, at which point the other judges are monitored; then if there are three judges or more pressing a button within INTERVAL, the model will award a score. Otherwise, the model resets all judges to the initial value where no one has pressed a button. However, in the following case, the model will fail to award a score:

When the judges press these buttons in the way shown in Figure 5.5, there will be three intervals and no score is recorded. However, this is abnormal because if we were to ignore judge 1, judges 3, 5 and 2 would correctly generate a score. This can be resolved by a more complex model (see Appendix D).

5.1.4 Verification

UPPAAL is able to check whether or not certain constraints on a clock or data variable satisfy the requirement of a system as well as certain locations are reachable from an initial location. The requirement must be expressed in a formally well-defined and machine readable language. Like Timed Computation Tree Logic, UPPAAL supports temporal properties in order to easily create requirements, classified *reachability*, *safety* and *liveness* properties. Let `p`, `q` be expressions which are formulae being either timing and data constraints or a location formed `process.location` where `process` is a timed automaton. There are five types of properties in UPPAAL [BDL04, UPP02].

Name	**Property**	**Equivalent to**	**Apply to**
Possibly	`E<>p`		Reachability
Invariantly	`A[]p`	not `E<>` not `p`	Safety
Potentially always	`E[]p`		Safety
Eventually	`A<>p`	not `E[]` not `p`	Liveness
Leads to	`p-->q`	`A[]`(`p` imply `A<>q`)	Liveness

Figure 5.6: Properties in UPPAAL

The property `E<>p` evaluates to true if there exists a path staring at an initial location satisfying a given `p`. It is useful to check possible behaviour of a system and is known reachability analysis. The properties are called reachability properties. For example, according to the Olympic Boxing Scoring system, if `E<> redScoreBoard.score == 1 and blueScoreBoard.score == 2 and Scheduler.oneRound` is satisfied, there exists a case in the system that the `redScoreBoard` and `blueScoreBorad` can reach 1 and 2 score respectively within one round.

The property `E[]p` is used to investigate whether there exists a path always satisfying `p`, and the property `A[]p` checks whether all paths always satisfy `p`. As described them with the word 'always', these two properties are useful to check safety of a system, such as they can examine 'something bad will never happen'. For example, using the property '`A[]`not deadlock', this property is able to make sure that a system will not be deadlock.

The property `A<>p` is to verify that all paths will eventually satisfy a given `p`. Similarly the property `p-->q` is to check not all paths but a specific path; it is read as whenever `p` is satisfied, and then eventually `q` will be satisfied. They are useful to examine liveness of a system as known liveness analysis. For instance, `Light.turnon->Light.bright` meaning that when pressing the button of a light, then the light eventually should turn on.

For the purpose of extending the properties, it is possible to transform them to equivalent properties such as `A[]p` is equivalent to not `E<>` not `p` and `A<>p` is equivalent to not `E[]` not `p`.

5.1.5 Model-Checking and State Explosion Problem

With the five properties provided in UPPAAL, it is easy to verify the correctness of a model (or system), known as model-checking. Model checking is typically based on an exhaustive state space search of a model, that is, each state of the model is examined whether the state satisfies the desired property correctly. In reachability analysis, model checking explores all reachable states in order to determine whether a model can reach to the state satisfying the property and no further state is possible.

However, one of the main obstacles in model checking is so-called state-explosion problem. Depending on a number of states in a model, the size of state space search will increase exponentially. For example, for an automaton having $m = |Q|$ control states and n merely boolean state variables, the model-checking will be starting at an automaton having $m \cdot 2^n$ states [AD90]. When a model particularly consists of many automatons $\mathcal{A}_1,\ldots,\mathcal{A}_n$ and synchronises each other, the size of the model is the order of $|\mathcal{A}_1| \times \cdots \times |\mathcal{A}_n|$ and thus the size of state space search will be exponential. In addition, it is known that the number of clocks and data variables particularly increase the states in a model.

It is recognised that the explosion problem depends on the explicit construction of the model targeted for verification. Consequently, when constructing a timed automata model for systems, the following questions must be kept in mind: Has a model been built explicitly? Are there any wasted clock or data variables? [BN03].

5.2 Scheduling Models with UPPAAL

Based on the principle that a task and message can be modelled by finite states and a clock variable, a job comprising a sequence of tasks and messages has been formalized by timed automata; the behaviour of an entire distributed system, thus, has been presented a parallel composition of these timed automata models. Using the UPPAAL tool, this section introduces a real timed automata model for the job and further creates a scheduler model to control system resource accessed by these models. In particular, there is a discussion with three UPPAAL models; each represents the model for a job in a slightly different way. This is not only to show the various ways to create a UPPAAL model for a job but also to find the most efficient model among them. A poor model may suffer from state-explosion problem in its verification and make model checking impractical. In order to explain each model, consider a simple job which consists of two tasks in processors and one message between the tasks in a network.

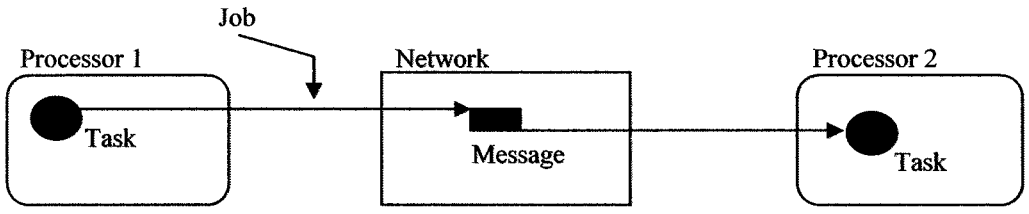


Figure 5.7: A Simple Job in Two Processors

5.2.1 Model One

First, the UPPAAL model consisting of seven locations and six transitions is considered for the simple job, shown in Figure 5.8. It has a clock, guards in some transitions, invariants in some locations, and many synchronisations using channels. The clock, guards and invariants are obviously used to check the computation time for a task and the transfer time for a message but the channels requires further explanation. The system being analysed may be simple – the system comprises two processors and one network and a simple job – but may have many jobs executed on it. In this case, several tasks in the same processor may perform and request the processor resource simultaneously even though it allows only one task to access. For this reason, a scheduler timed automata model is introduced (see Figure 5.8) in term of controlling a

processor resource. With the scheduler model, it is possible to allow only one task access and other tasks in the same processor will have to wait until its occupation is finished. Each processor and network in a system is required to have a scheduler model, and thus, when considering the simple job, it is necessary to create three schedulers: one for each of the two processors and one for the network.

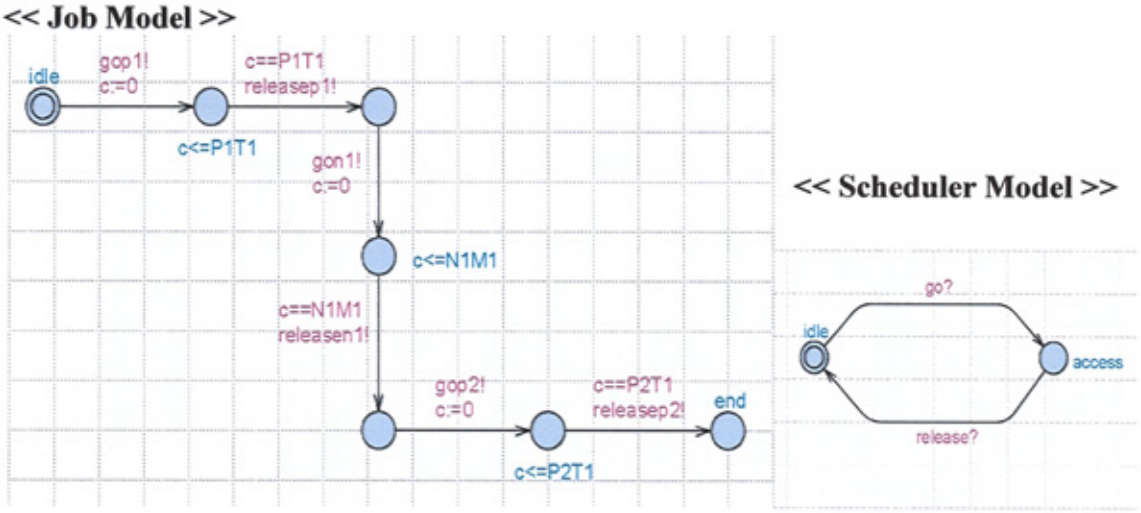


Figure 5.8: Job and Scheduler Models in Model One

The working principle of the simple job model is simply explained by following the transitions of the model. When the job wants to execute its first task, it requests access of the corresponding processor by synchronisation via a channel to the scheduler for the processor. If the scheduler is waiting the synchronisation (the scheduler is an idle location), the job will be given the access of the processor otherwise it will wait until the access is released. The channel has to be an urgent channel because this prevents any synchronisation delay and also gives only one entry possible to access during the synchronisation. When the job completes its first task, it signals its completion to the corresponding scheduler by a channel. This procedure will be continued until the job meets its end. Importantly, the start and finish of each task and message in the job is decided by a clock variable which is used to record their computation times and transfer times.

This model can be easily extended with more tasks and messages. The number of the locations in the model is expected $n(J_i)\times 2+1$ where $n(J_i)$ is a function which

indicates a total count of the tasks and messages in J_i and the number of the transitions is $n(J_i)\times 2$. As each job model in a system requires one clock, the total number of clocks used in creating the entire models for the system will be the same as the number of jobs such denoted by $n(\mathcal{C}) = n(\mathcal{J})$.

5.2.2 Model Two

According to [BBF+01, LPY95a, LPY95b], the number of clock variables affects the complexity of a system; in particular, it is one of the causes of the state-explosion problem during model-checking. In a case of a system using Model One (see Figure 5.8), for example, if the system has many jobs, the number of clocks will be increased as the same as the number of jobs. So, it may suffer from the explosion problem in the system with many jobs. As a result of this, Model Two is introduced as a means of reducing the number of clock variables. Based on the Model One, small changes are applied in Model Two such that the clock variable used in the Model One moves to the scheduler model. This means that to evaluate the execution time of a tasks and message is possible in the scheduler model rather than the job model. Thus, the number of clocks in a system will not be increased whatever jobs are created in the Model Two.

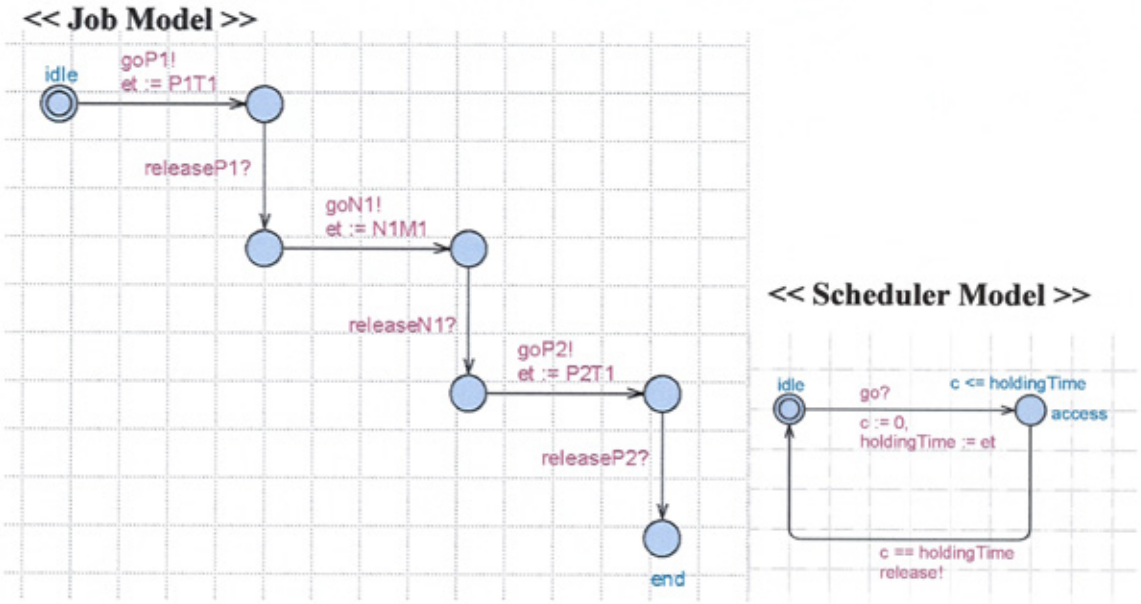


Figure 5.9: Job and Scheduler Models in Model Two

Basic operation is similar to the Model One. When a job wants to start its task or message, it requests access to the corresponding scheduler via a channel. But, this time it sends not only synchronisation but also gives the execution time of its task or message via a global variable. Thus, the scheduler can automatically refer to the execution time from the variable. Once the scheduler starts with resetting the clock variable with zero, it holds its access only for the time passed by the global variable. Since then, it will send back the synchronisation to the job model in order to signal the end of the execution.

This model only has some changes on the existing locations and transition in the Model One. Thus, the number of the locations and transitions in Model Two will be the same as the Model One, $n(J_i)\times 2+1$ and $n(J_i)\times 2$ respectively. However, the total number of clocks in a system using the Model Two will be different from Model One, because it is not any more dependent on the number of jobs but on the number of processors and networks in a system. Consequently, the total number of clocks is expected to be $n(\mathcal{P})+n(\mathcal{N})$.

5.2.3 Model Three

Model Three is introduced to minimise locations in the model because the number of locations might also affect the complexity of a system during model-checking. This model is used to find whether there is connection between the number of locations and guards, and state spaces in model-checking. According to both the job models introduced above, the number of locations in the models depends on the size of a job, in other words, the number of tasks and messages belonging to the job. For example, in a case where a job includes many tasks and messages, the number of locations in both the job models will inevitably be larger. As a result, Model Three is introduced as a means of reducing the number of locations in the models. Based on the Model Two, the Model Three forces each task and message in a job to share locations by adding a guard in the model.

Model Three works on the guards added onto each start transition of a task and message which indicate conditions suited for the execution order of a job. Given the integer variable, `order`, assigned the value 1 at first, the model can only take a course satisfying the condition that the variable is equal to 1. When any given guard in the

model is satisfied, it requests an access of the corresponding scheduler via a channel. Once the scheduler is completed, the job model returns to the `idle` location and also changes the variable in order to make the guard of the succeeding work be satisfied such as increasing the variable by 1. This procedure will be continued until the job model reaches the `end` location which represents the end of the model. As a result, the `idle` location can be used as a shared location among the other tasks and messages in the job model. The Model Three is shown in Figure 5.10.

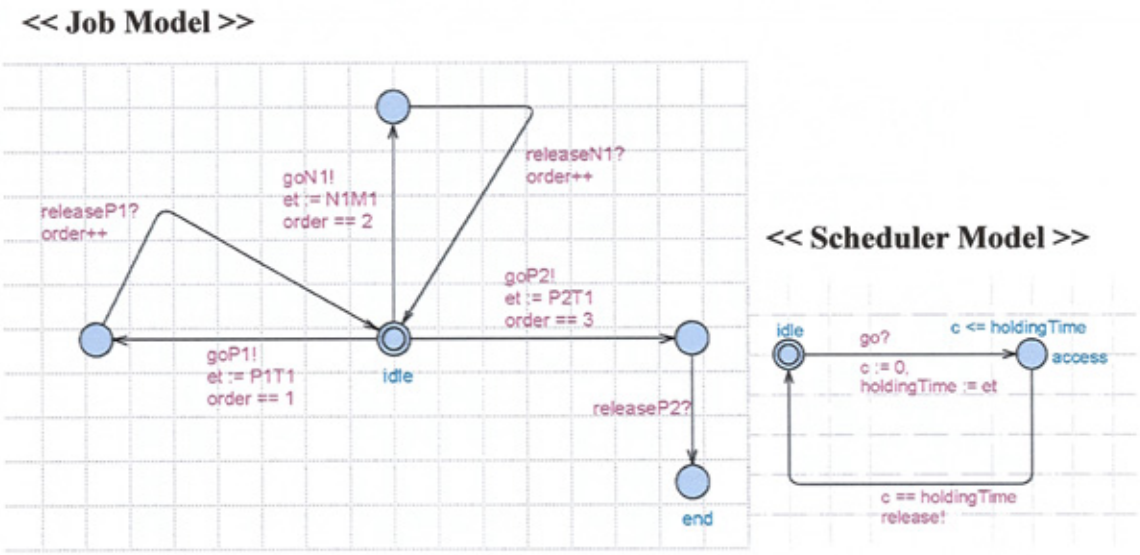


Figure 5.10: Job and Scheduler Models in Model Three

Because Model Three is based on the Model Two, the number of transitions and clocks are expected as the same as the Model Two which are $n(J_i)\times 2$ and $n(\mathcal{P})+n(\mathcal{N})$ respectively. The number of locations is different. In the Model Three, the `idle` location can be shared by the others tasks and messages. So, it is possible to make each task and message by only one location, except for the last task in a job. Accordingly, the number of location in the model is $n(J_i)+2$. However, there is a trade-off. This model can reduce the number of locations but unfortunately not avoid creating guards on transitions. Table 5.1 summaries the number of locations, transitions and clocks used in the three models.

Model	Locations	Transitions	Clocks
Model One	$n(J_i)\times 2+1$	$n(J_i)\times 2$	$n(\mathcal{J})$
Model Two	$n(J_i)\times 2+1$	$n(J_i)\times 2$	$n(\mathcal{P})+n(\mathcal{N})$
Model Three	$n(J_i)+2$	$n(J_i)\times 2$	$n(\mathcal{P})+n(\mathcal{N})$

Table 5.1: Number of Locations, Transition and Clocks in the Three Models

5.3 Evaluation of the Models

Three different models for a job have been introduced with some differences between them such as the number of clocks and locations. It is expected that the differences certainly affect their performance during model checking because a poor model may suffer from the state explosion problem. At this point, it is valuable to find the most efficient model among them. The models have been evaluated and consequently the most efficient model among them has been discussed below.

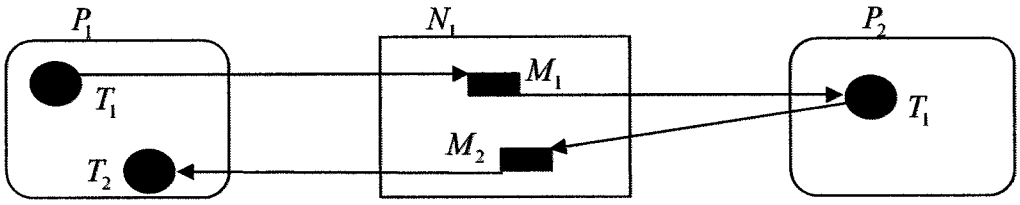


Figure 5.11: A Job for Evaluation on Two Processors

Consider an extended job which is a similar one shown in the Figure 5.7 but has more tasks and messages. It is supposed that there is a system consisting of two different processors via a network, and that there is the job as a typical control job in the system. The job generally works as follows: it collects data and then sends the data to other task in other processor by message passing. The task receiving the data performs some control works in order to compute an appropriate control output. When having completed its execution, it returns the control output to a task in the original processor by a message. The job is depicted in Figure 5.11.

When creating the UPPAAL model for the job based on the Model One, it is expected that the model will have 11 locations, 10 transitions and 1 clock according to

the Table 5.1 which represents the number of transitions, locations and clocks for the three different models. Also, based on the Model Two, the model will consists of 11 locations, 10 transitions and 3 clocks; the Model Three will require 7 locations, 10 transition and 3 clocks. Figure 5.11 shows the UPPAAL model based on the Model One.

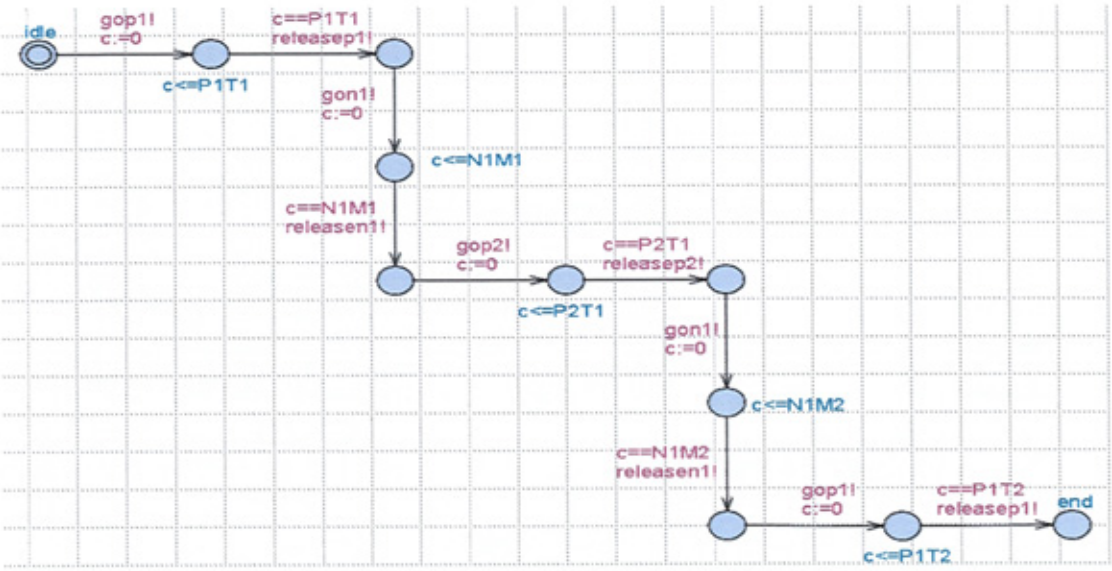


Figure 5.12: The UPPAAL Model for the Extended Job based on the Model One

With the UPPAAL models for the extend job based on the three different models, the system increases the number of jobs one by one in order to vary their complexity. This is to clearly distinguish which model suffers more from the state explosion problem during their verification. However, the increase has to be limited to avoid severe explosion problem caused by adding too many jobs.

Using the application, *verifyta*, supported in the UPPAAL tool working on the computer environment with CPU 2.20GHz and 512MB of RAM on Windows OS, the time consumption is measured until the system reaches the `end` location in each model with different number of jobs. Starting with two jobs, the number of jobs is increased up to 7 jobs. It is assumed that all the tasks and messages are 10 times units for their computation times and transfer times. Also, different properties are required depending on the number of jobs. For example, when considering 2 jobs within 100 time-units, the property used for the measurement was `E<>j1.end and j2.end and c < 100` where `j1` and `j2` are the names for jobs `and` c is a clock variable, meaning that the property

looks for any path satisfying the `j1` and `j2` reach the `end` location within 100 time-units. Table 5.2 shows the result of time consumption required by the three models.

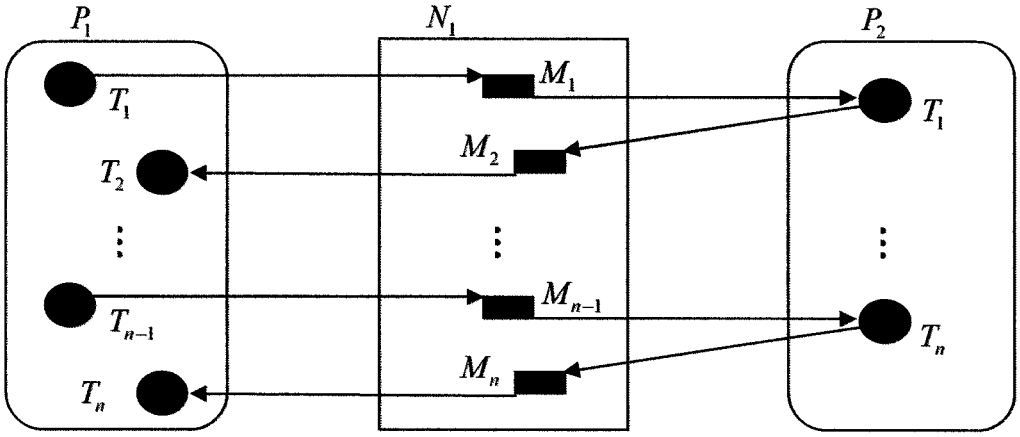


Figure 5.13: The Number of Extended Jobs in a System

Unit: milliseconds

	Model One	Model Two	Model Three
2 Jobs	30	20	30
3 jobs	40	40	40
4 jobs	150	100	150
5 jobs	1412	841	1031
6 jobs	14070	7631	9994
7 jobs	130318	73305	92473

Table 5.2: The Time Consumption for the Models

According to the result, the following factors are recognised:

- The differences in time consumption of the three models appear small for just a few jobs, but the differences become more apparent as the models get larger. The result shows that with 2 and 3 jobs, the time consumptions are almost the same in the models but with more than 3 jobs, the differences are larger and even more dramatic with 7 Jobs (See Figure 5.14). It means that the more the models suffer from explosion problem by increasing jobs, the easier the time consumption of models are distinguished. However, all the models suffer from the state explosion problem after 7 jobs.

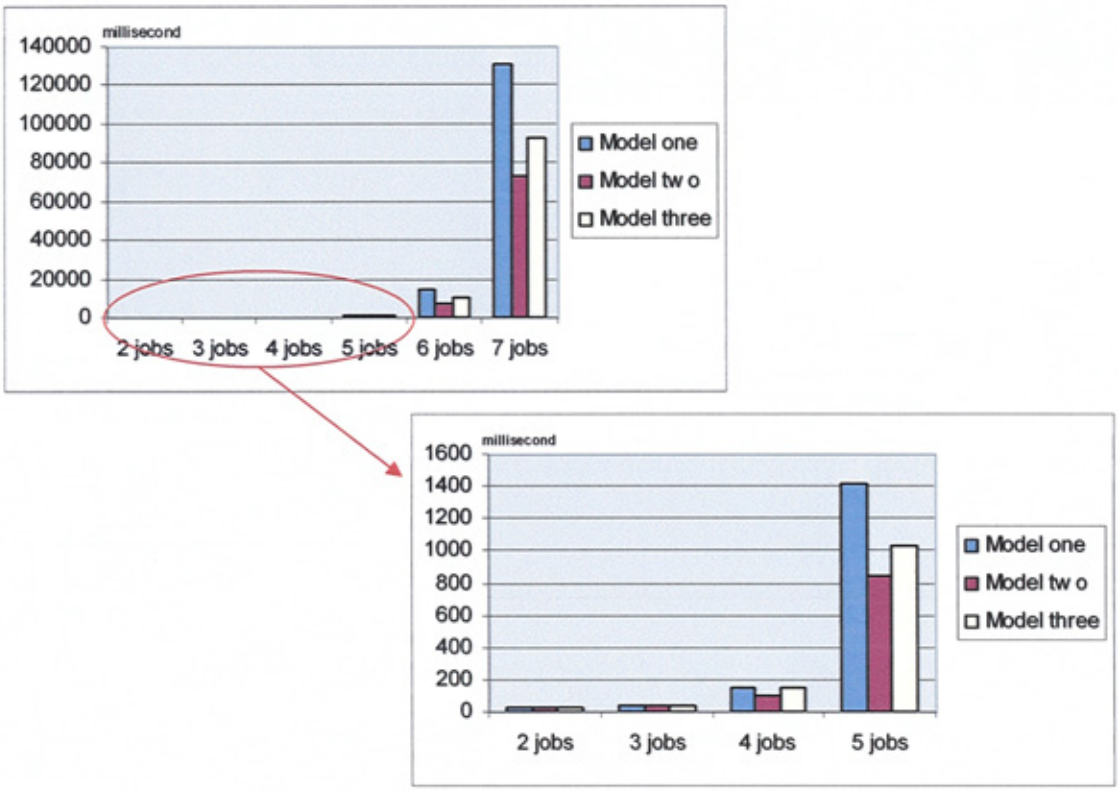


Figure 5.14: The Graph for the Time Consumption for the Models

- The Model Two takes the least time in most cases but Model One is the greatest. This is obviously suspected that these two models have a different number of clock variables. In Model Two, each processor and network includes one clock variable in order to handle their resource accessibility; it needs always 3 clock variables without considering the number of jobs. However, Model One needs one clock variable depending on the number of jobs; it goes up to 7 clock variables. With 2 and 3 jobs, there is almost no time difference because the two models have a comparable number of clock variables used and also may be not complex. After then, it shows considerable time difference. In particular, the difference is almost twice with 7 jobs (See Figure 5.14).
- The Model Two also takes less time consumption than Model Three even though both of the models have the same number of clock variables. It is expected that the difference comes from a different number of locations and guards between the two models. Model Three has a reduced number of locations

because tasks and messages share some location. But, inevitably it has more guards on transitions in order to create its execution order. According to the graph, the result shows that the number of guards has more influence than the number of locations during verification. However, the Model Three takes less time than Model One as it employs fewer clocks.

The system in this evaluation, however, is only an example. It is possible to create different cases, such as a system with many processors and networks but few jobs. In this case, Model One will use few clock variables but Model Two will have many. And then the comparison result will appear to be reversed. Overall, it is understood that a clock variable is an important factor during model checking as the number of clocks used in creating models affects the state explosion problem. Thus, it is required to choose an efficient model in order to reduce clock variables to as few as possible. In particular, considering designing schedules of systems using the models, it is recommended that the Model One is the better choice in a case that a system has many processors and networks. Otherwise, the Model Two is the better one. However, state explosion problem is unavoidable when a system gets complex. This is why high performance machines are required during model checking.

5.4 The Limitation of the Timed Automata Model

As the behaviour of an entire system is a parallel composition of timed automata models as a set of jobs, the size of an entire system model will be dependent on the number of jobs. When a system has more jobs, the system model will be more complicated and eventually suffer from the state explosion problem in its verification.

This has a significant effect to the model scalability. In order to investigate this issue, the time consumption of a system model in verification is measured when gradually increasing the number of jobs in the system. The testing environment used to measure the time consumption is based on Windows Operating System with 512M memory and Celeron CPU 2.20GHz. Also, the Model Two model is used to express a job since it gives the best verification performance compared to the other models shown in the previous section.

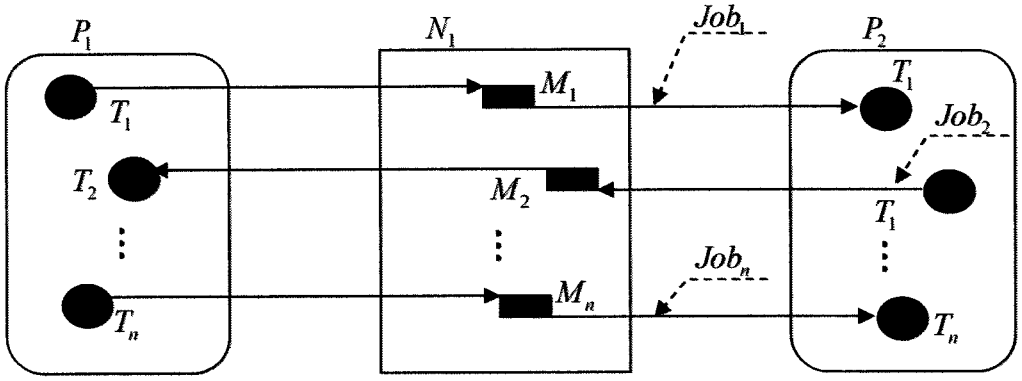


Figure 5.15: The System used to Check State Explosion Problem

Consider a system which has two processors connected by a communication network and a job comprising two tasks in the processors and one message between the tasks in the network. It is assumed that all the tasks and messages are 10 time-units for their computation times and transfer times, and the period of all jobs are 200 time-units. The system has only one job at first. The time consumption for the system verification is measured using a property which simply checks the existence of a schedule. For example, `E<>j1.end and c < 200` is used when considering a schedule with 1 job within 200 time-units, meaning that this property looks for any path satisfying the `j1` reaches the `end` location within 200 time-units. The number of jobs is increased by one at a time as shown in Figure 5.15. Time consumption at each stage is measured by a corresponding property which depends on the number of jobs. Table 5.3 shows the results.

Unit: milliseconds

Jobs	1	2	3	4	5	6	7	8	9	10
Time	30	40	40	60	140	641	3084	17756	150557	-

-: Memory overflow with more than three hours of execution time

Table 5.3: The Time Consumption with the Different Number of Jobs

The differences in time consumption become disproportionably large as the system gets more jobs. The result shows that up to 8 jobs, the computation times in their

verification are acceptable as the times are less than 18 seconds. But, it takes 150 seconds to get a result with 9 jobs. Even with 10 jobs, it fails to reach to the end of its verification as the testing environment can not support enough memory and also it takes more than 3 hours to get the memory overflow. It means that on this testing environment, a system, which has less than 9 jobs, will be considerable working with timed automata models. However, this result will be different when considering different type of jobs and different testing environment, but it is inevitable from the sate explosion problem. It is difficult to consider many jobs in a system and thus medium size of system will be tractable due to the explosion problem.

5.5 Summary

This chapter has introduced UPPAAL which is a timed automata tool for symbolic model checking. Some notations employed in UPPAAL models and the properties used in verifying the models have been summarised with the Olympic Boxing Scoring system applied to the UPPAAL. Based on the formalisation of a job by timed automata, the required UPPAAL model has been created. Further, in order to ameliorate the state explosion problem of the UPPAAL model, three different models have been examined for their efficiency. It is understood that a clock variable is an important factor during model checking and the Model Two amongst the three models is the best choice, in a case that a system has many jobs, i.e. it uses fewer clocks. However, the state explosion problem is inevitable when considering a complicated system which has many jobs.

Chapter 6

Scheduling Generation with UPPAAL Models

With the UPPAAL models representing the behaviour of distributed real-time system in a time-triggered architecture, this chapter proposes a novel approach to generating feasible schedules using these models and further introduces a prototype toolset based on the approach. In order to describe the approach and the prototype toolset, a simple but representative Fluid Control system is analysed to derive schedules.

6.1 The Proposed Approach

As explained in the previous chapter, UPPAAL is a timed automata tool for symbolic model checking, supporting formal modelling, verification and simulation. This tool, in particular, has a useful facility which enables the generation of a trace file as the result of verification or simulation. The file is produced by either using the simulator in the tool (manually choosing each enabled transition in each state until a satisfied result is reached) or using the verifier (automatically choosing transitions and states until a given property is satisfied in diagnostic verification). In the case of the latter, UPPAAL further provides the stand-alone command called *verifyta*. With this UPPAAL facility of generating a trace file an efficient approach is proposed to generate schedules for distributed real-time systems in a time-triggered architecture.

The proposed approach is that given UPPAAL models representing the behaviour of a system, i.e. a parallel composition of job and scheduler models shown earlier, it is possible to verify the models with a temporal logic property adequately representing the system specification. And then, if the property is satisfied with the given model in their behaviour verification, the UPPAAL verifier is forced to produce a diagnostic trace as a

successful example of the verification. The trace consequently produced by the verifier will be simply analysed to automatically generate a feasible schedule for the system. Figure 6.1 shows a pseudo code algorithm for the approach.

```
...

Let M, TCP, TF be UPPAAL models, a property and a trace file
respectively. They are initially all empty

for each P_i in P
    M = M ∪ CreateUPPAALModel(P_i)    // Creating Models For Processors
end for

for each N_i in N
    M = M ∪ CreateUPPAALModel(N_i)    // Creating Models For Networks
end for

for each J_i in J
    M = M ∪ CreateUPPAALModel(J_i)    // Creating Models For Jobs
end for

Let isSatisfied ∈ {true, false} be initially false

do while isSatisfied
    TCP = CreateProperty(M);          // Creating a Property
    if M |= TCP then
        TF = GenerateTrace(M, TCP)    // Creating a Trace
        isSatisfied = true
    end if
end do

return TF

...
```

Figure 6.1: Pseudo Code Algorithm for the Proposed Approach

The UPPAAL verifier performs an exhaustive state-space search in verification [BBD+02, LPY97], meaning that it examines every possible state of the models in order to find a satisfaction for a given temporal property. The proposed schedule generation approach employing the verifier is also based on the same search to find a feasible schedule for a system. Although there is an issue of the state explosion problem in the

approach, particularly in cases of exploring complex and large systems, the search will contribute important decisions for schedule generation. The approach has the following characteristics:

- If a schedule exists for a system, the approach must find the schedule.
- If a schedule is not found in the approach, there is no schedule for a system, thus if a schedule does not exist for a system, the approach can identify its non-existence
- If a schedule is identified in the approach, the schedule must be correct.

Thus, this approach guarantees to generate a practical schedule if one exists and will fail to construct any schedule if none exists. Also, it guarantees to identify the existence of a schedule. In particular, this modelling approach is radically different from the existing methods or algorithms for global scheduling which are based on the complicated mathematical approaches such as heuristic methods, simulated annealing, genetic algorithms, etc. Consequently, this approach provides not only a guarantee for scheduling generation but also more general, understandable, and applicable functionalities than alternatives.

However, there are difficulties in manually applying the proposed approach for the following reasons:

- **Creation**: Creating UPPAAL models based on a system $\mathcal{S}=(\mathcal{P},\mathcal{N},\mathcal{J})$, or including additional models into the existing models for a system, requires considerable manual effort and care in practical views.
- **Analysis**: The trace file cannot be used directly to generate schedules without further processing because of the complexity of the transition and state information in it. Thus, it is required to abstract data appropriate to the purpose of the approach from the file.
- **Visualisation and Extraction**: It is difficult to visualise schedules. With the proper data abstracted from the file, it would be useful to graphically present schedules and further to extract schedules for each processor and each network.

In order to overcome these difficulties in the approach, it is necessary to provide such a standard way which includes automatic procedures to generate schedules for systems by creating UPPAAL models, analysing a trace file, visualising and extracting schedules. Therefore, the prototype toolset is introduced; Figure 6.2 provides an overview of the toolset. There are two phases involved in it: Preprocessor and Analyser Engine. The Preprocessor provides the way of describing a system specification with XML, including the behaviours and constraints such as processors, tasks, messages, jobs, timing constraints, etc. With the system specification expressed in XML, the Analyser Engine phase involves in the following procedures:

1. Generate the timed automata models for processors, networks and jobs, and the temporal logic property representing the system specification.
2. Execute the UPPAAL verifier with the automata models and the property, and generate a trace file as the result of their successful verification.
3. Analyse the generated trace file in order to abstract schedules for the systems.
4. Display the schedules graphically.
5. Extract schedules for each processor and each network.

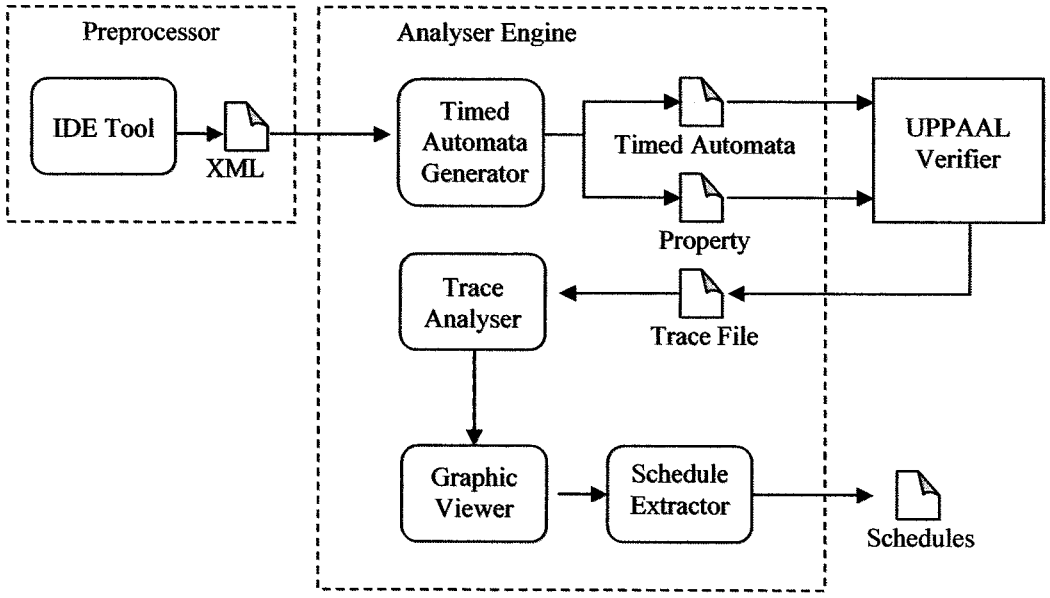


Figure 6.2: Overview of the Approach with the Prototype Toolset

6.2 Prototype Toolset Based on the Approach

A simple but representative scenario is presented to give an understanding of the approach and prototype toolset. Considering a Fluid Control system mentioned in an earlier chapter, as an example of distributed real-time system using a time-triggered architecture, the approach is applied to the system using the prototype toolset in order to explain each function in the toolset.

6.2.1 Fluid Control System

Supposing that a Fluid Control system (called the FC system) requires two processors: `Plant` and `Consol`, connected by a network (see Figure 6.3). The `Plant` processor has three tasks periodically triggered: `Sample`, `Actuator` and `Alarm` tasks. There are two tasks on the `Consol` processor: `Controller` and `Indicator` tasks. These tasks communicate values by messages passing on the network; there are three messages involved in the communication named `Pressure`, `Alarm` and `Valve` messages, as indicated in Figure 6.3.

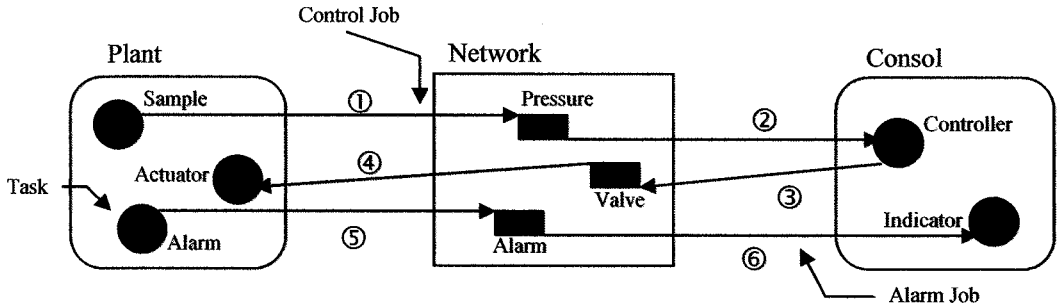


Figure 6.3: The Fluid Control System

The system contains two jobs: `Control` and `Alarm`, which are used to check the rate of flow and to inform alarm information to the `Consol` processor respectively. The `Control` job consists of three tasks engaged with two messages and has the following procedures:

①. The `Sample` task in the `Plant` processor is activated as the start of the job, and when finishing its work it sends a message (`Pressure`) on the network.

② The Pressure message has to be sent from the Plant processor to the Consol processor via the network.

③ The Controller task invoked periodically reads the Pressure message sent and gives a feedback to the Plant processor by sending the Valve message.

④ The Actuator task is executed and reads the Valve message.

Similarly, the Alarm job adopts almost the same procedure as the Control job but is simpler, working with only two tasks and one message:

⑤ The Alarm task in the Plant processor checks alarm information and then sends it via the Alarm message.

⑥ The Indicator task in the Consol processor displays the information.

This chapter will present a simpler explanation of the proposed approach using uncomplicated timing values allocated to the tasks and message in the system. Thus, it is simply assumed that the period of the Control job is 100 time-units and the Alarm job is 50 time-units and further assumed that the computation time of all tasks in the system is 10 time-units and the transfer time of all messages is 10 time-units (see Table 6.1).

Job	Task/Message Information				
Control Job	**Task**	**Period**	**Computation**	**PC_In**	**PC_Out**
	Sample	100	10	N/A	Pressure
	Controller	100	10	Pressure	Valve
	Actuator	100	10	Valve	N/A
	Message	**Period**	**Transfer**	**PC_In**	**PC_Out**
	Pressure	100	10	Sample	Controller
	Valve	100	10	Controller	Actuator
Alarm Job	**Task**	**Period**	**Computation**	**PC_In**	**PC_Out**
	Alarm	50	10	N/A	Alarm
	Indicator	50	10	Alarm	N/A
	Message	**Period**	**Transfer**	**PC_In**	**PC_Out**
	Alarm	50	10	Alarm	Indicator

Table 6.1: The Allocated Values in the FC System

6.2.2 Preprocessor

Creating models in the UPPAAL tool is laborious work despite its convenient graphical interface. Required is not only the knowledge of using the tool but also the knowledge of related subjects such as model checking, temporal logic, timed automata etc. It is such time consuming work whether models are created or updated. For this reason, a standard way of expressing the given specification of systems using XML is provided, which is independent of the UPPAAL tool. XML is as a subset of SGML (*Standard Generalized Markup Language*). It gives efficient creation, maintenance and storage of information. And so it can be re-used and re-composed as need requires [Rat98]. The standard way of the expression is provided, based on the terms defined previously:

- System $S=(\mathcal{P},\mathcal{N},\mathcal{J})$
- Processor $P_i=(T_i)$
- Task $t_i=(p,c)$
- Network $N_i=(M_i)$
- Message $m_i=(p,\pi)$
- Job $J_i=(p_i,w_0,w_n)$

The following sections introduce how each of these terms can be described using XML without losing their native meaning.

6.2.2.1 Description of a System with XML

First, the system term which is the root of the remaining terms is defined. It is remembered that a system is defined as $\mathcal{S}=(\mathcal{P},\mathcal{N},\mathcal{J})$ comprising the set of processors, networks and jobs. On the basis of the definition, a system is described in XML as follows:

- It is formed with starting `<tt_system>` tag and finishing `</tt_system>` tag.
- The `<systemID>` and `</systemID>` tags are required to display the name of a system.

- It includes <processor> and </processor>, <network> and </network> and <job> and </job> tags which define each processor, network and job respectively.
- Multiple definitions for the set of processors, networks and jobs are expected.

Figure 6.4 shows the structure of a system with XML description of the FC system; Fluid Control System is defined as the value of <systemID>. The Figure omits other definitions but the following sections below introduce further detail. The whole XML description is presented in Appendix B.

$S=(\mathcal{P},\mathcal{N},\mathcal{J})$	<tt_system> <systemID> Fluid Control System </systemID> <processor> ... </processor> ... <network> ... </network> ... <job> ... </job> ... </tt_system>

Figure 6.4: Description of a System with XML

6.2.2.2 Description of a Processor with XML

A processor $P_i \in \mathcal{P}$ is associated with a set of tasks, T_i, where $T_i \subseteq T$ and T is all tasks in S and each task in the set is defined as a tuple $t_i = (p,c)$ which comprises the period and computation time of a task. With the definitions, a processor in XML is defined:

- It is formed by starting with the <processor> tag and finishing with the </processor> tag.
- The <processorID> and </processorID> tags are required to represent the name of a processor; the name has to be a unique value in order to distinguish it from other processors.
- The <task> and </task> tags are used to express the tuple for each task.

- The name of a task is described by the <taskID> and </taskID> tags; the value has to be unique in order to distinguish it among other tasks.
- The <period> and </period> tags are required to describe the period of a task periodically invoked.
- Two pair tags, <pc_in> and </pc_in>, and <pc_out> and </pc_out> indicate the predecessor and successor of a task respectively.
- The <computation> and </computation> tags are required to indicate the worse case computation time of a task.

• Many descriptions for a task may be required as a processor has a set of tasks.

Figure 6.5 shows the way of describing a processor with XML, in particular, using the Plant processor in the FC system. And it also includes the Sample task which is executed on the processor. As previously indicated the values for the Sample task in Table 6.1, the task is described with 100 time-units of the period, 10 time-units of the computation time, and Environment and Pressure as the precedence constraints.

$P_i = (T_i)$ $t_i = (p, c)$	... <processor> <processorID> Plant </processorID> <task> <taskID> Sample </taskID> <period> 100 </period> <computation> 10 </computation> <pc_in> Environment </pc_in> <pc_out> Pressure </pc_out> </task> <task> ... </task> ... </processor> ...

Figure 6.5: Description of a Processor with XML

6.2.2.3 Description of a Network with XML

A network $N_i \in \mathcal{N}$ has a number of messages, M_i, where $M_i \subseteq M$ and M_i is all messages in S and each message is said as a tuple $m_i = (p, \pi)$ including the period and the time for transferring the message. The following shows how to describe a network in XML:

$N_i = (M_i)$ $m_i = (p, \pi)$	... <network> <networkID> Ttp </networkID> <message> <messageID> Pressure </messageID> <period> 100 </period> <transfer_time> 10 </transfer_time> <pc_in> Sample </pc_in> <pc_out> Controller </pc_out> </message> <message> ... </message> ... </network> ...

Figure 6.6: Description of a Network with XML

- It is formed by starting with the <network> tag and finishing with the </network> tag.
- The <networkID> and </networkID> tags are required to represent the name of a network, which has to be a unique value to distinguish it from other networks.
- It contains the set of messages and each is represented with <message> and </message> tags.
  - A unique value is expected by describing the <messageID> and the </messageID> tags to distinguish it among other tasks.
  - The <period> and </period> tags are required to describe the period of a message periodically invoked.
  - Two pair tags, <pc_in> and </pc_in> tags, and <pc_out> and </pc_out> tags describe the predecessor and successor of a message respectively.
  - The <transfer_time> and </transfer_time> tags are used to indicate the transferring time of a message.

Figure 6.6 shows the structure of a network describing the network in the FC system. The network is called Ttp and has a couple of messages. One of the messages, Pressure, has 10 time-units to transfer it; it is transmitted every 100 time-units. According to the Figure, it is clear that the predecessor of Pressure message is Sample task and the successor is Controller task.

6.2.2.4 Description of a Job with XML

It is known that a job as a finite sequence of tasks and messages is defined a tuple $J_i = (p_i, w_0, w_n)$ which includes the period of the job and indicates the start task and end task of the job respectively. It is assumed that the precedence constraints between tasks and messages, which are defined by w_0 to w_n, are correct such that $w_i \bullet \underrightarrow{mb} = w_{i+1} \bullet \overrightarrow{mb}$ for $0 \leq i < n$. The description of a job is in XML as follows:

- It is formed by starting with the `<job>` tag and finishing with the `</job>` tag.
- The `<jobID>` and `</jobID>` tags are required to represent the name of a job and the name value has to be unique.
- The start task in a job is described by `<from>` and `</from>` tags; the sub-tags `<taskID>` and `</taskID>`, and `<processorID>` and `</processorID>` are employed to describe its further information.
- The end task is described by `<to>` and `</to>` tags; the sub-tags `<taskID>` and `</taskID>`, and `<processorID>` and `</processorID>` describe its further information.
- The `<within>` and `</within>` tags are used to indicate the period of a job.

$J_i = (p_i, w_0, w_n)$	... <job> <jobID> control_job </jobID> <from> <taskID> sample </taskID> <processorID> plant </processorID> </from> <to> <taskID> actuator </taskID> <processorID> plant </processorID> </to> <within> 100 </within> </job> ...

Figure 6.7: Description of a Job with XML

Figure 6.7 describes one of the two jobs in the FC system, `Control` job, with XML, which is named as `control_job` and the job involves three tasks and two messages

according to the Table 6.1. The description for the job in XML includes the following information such that it starts from the `sample` task in the `Plant` processor and ends in the `actuator` task in the same processor; the job has 100 time-units of the period.

6.2.2.5 Description of the FC System with XML in the Tool

It is possible to describe the FC system using the above XML descriptions of the terms; appearing in full in Appendix B. Although the XML for the FC system might be long, it fully and unambiguously defines the description of systems. At the moment, the method generating XML descriptions is via either directly typing them in the XML window provided in the toolset or indirectly loading a file already prepared using text-edit tools such as `NotePad` or `WordPad` windows applications. However, it will be expected to introduce an efficient IDE functionality in the toolset in future and so it will be able to reduce mistakes in typing and provide greater convenience in expressing them.

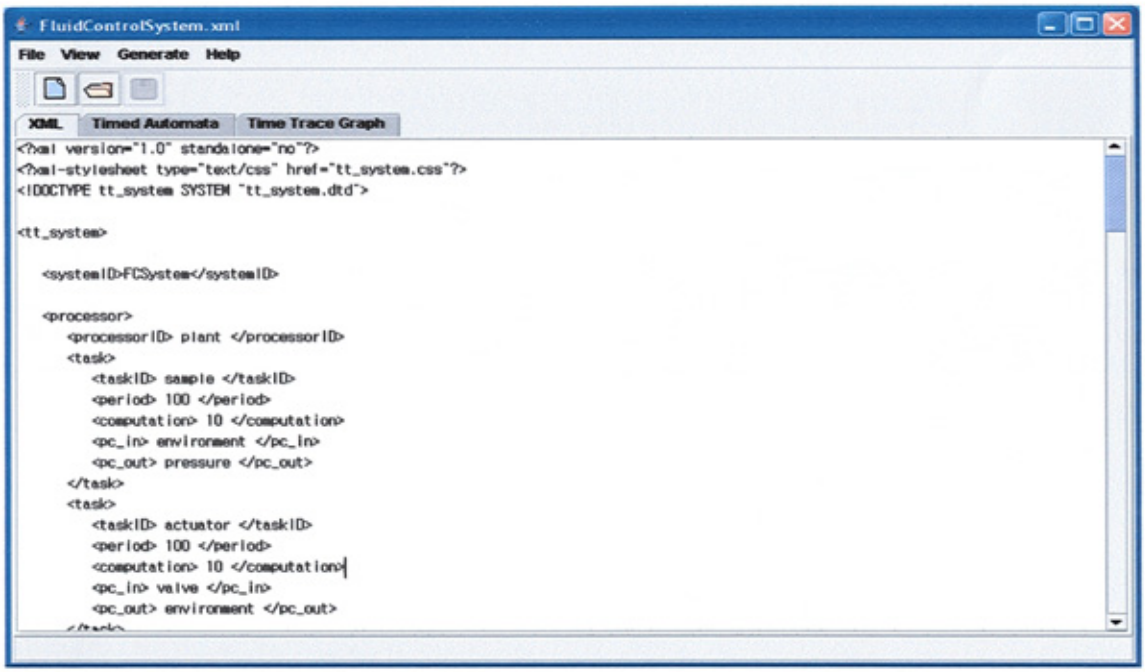


Figure 6.8: XML Window in the Toolset

6.2.3 Analyser Engine

With systems specified in XML, the Analyser Engine phase includes the following functionalities in the toolset: Timed Automata Generator, Trace Analyser, Schedule

Extractor and Graphics Viewer. Each of these functionalities is invoked in this order and is a part of producing the schedule for the given system specification. Each of those is outlined in detail.

6.2.3.1 Timed Automata Generator

The Timed Automata Generator, as the name implies, generates timed automata models according to the given system specification expressed in XML and thus generate automata models capable of working in the UPPAAL tool. As already described in early chapters, the tool provides two types of formats in expressing automata models, which are either a graphical format using the graphical interface or a textual format using the description language. The graphical format has the benefits of supporting a user-friendly interface, a convenient way of creating models and particularly providing straightforward awareness of models. However, the textual format is preferred in the toolset for the following reasons: the textual format provides a basic programming language and also provides faster procedures than the graphical format [LPY97] because internally the graphical format is transformed into the textual format when being used in verification; also the toolset uses the verification function by directly calling the stand-alone command to verify models.

After analysing the given system specification in XML, the Generator produces the textual format of models for each job, processor and network, and also a property for verification. With the `Control` job in the FC system, the textual format for the job is described first.

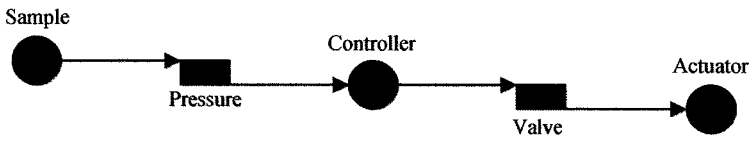


Figure 6.9: The Control Job in the FC System

All three tasks and two messages in the `Control` job (see Figure 6.3 or 6.9) are transformed into 11 states which is including all the initial states and final states for them. According to the syntax of the textual format in UPPAAL tool, all the states can be described using `state` and also indicate the required computation time and transfer time for each task and message using `{ }` as an invariant. For example, when

presenting the `pressure` message in the job, two states are involved `pressure_2`, `pressure_3` and are defined with the invariant of the message as `pressure_2, pressure_3{c <= 10}`. See the whole states with the invariants for the `Control` job defined below:

```
...
state idle, sample_1{c <= 10}, pressure_2, pressure_3{c <= 10},
controller_4, controller_5{c <= 10}, valve_6, valve_7{c <= 10},
actuator_8, actuator_9{c <= 10}, end{t <= 100};
...
```

The `Control` job also has 11 transitions to specify the precedence constraints between the tasks and the messages, including a transition to represent the periodic repetition of this job. Starting by `trans` syntax, a transition between two states is defined using `->` symbol. For example, the transition between `pressure_2` and `pressure_3` is defined as `pressure_2 -> pressure_3`. Furthermore, more properties on a transition are described inside `{ }` such as a guard by `guard` syntax, a synchronisation by `sync` syntax and an assignment by `assign` syntax. When describing properties on the transition between `pressure_2` and `pressure_3` with the synchronisation variable `gon0!` and the assignment 0 to clock *c*, it is consequently defined as `pressure_2 -> pressure_3{sync gon0!; assign c := 0;}`. More transitions are added, followed by ',' at the end. See the whole transitions of the `Control` job below:

```
...
trans idle -> sample_1{sync gop0!; assign c := 0;},
sample_1 -> pressure_2{guard c == 10; sync releasep0!; assign
hasrun := 1;},
pressure_2 -> pressure_3{sync gon0!; assign c := 0;},
pressure_3 -> controller_4{guard c == 10; sync releasen0!;},
controller_4 -> controller_5{sync gop1!; assign c := 0;},
controller_5 -> valve_6{guard c == 10; sync releasep1!;},
valve_6 -> valve_7{sync gon0!; assign c := 0;},
valve_7 -> actuator_8{guard c == 10; sync releasen0!;},
actuator_8 -> actuator_9{sync gop0!; assign c := 0;},
actuator_9 -> end{guard c == 10; sync releasep0!;},
end -> idle{guard t == 100; assign t := 0;};
...
```

Although there is more syntax to explain the textual format for the `Control` job such as `clock`, `int`, `init`, etc, it is recommended to refer to the following references [BY04, LPY97]. Figure 6.10 shows the complete textual format for the `Control` job.

```
process Rcontrol_job {
clock c, t;
int [0, 1] hasrun;

state idle, sample_1{c <= 10}, pressure_2, pressure_3{c <= 10},
controller_4, controller_5{c <= 10}, valve_6, valve_7{c <= 10},
actuator_8, actuator_9{c <= 10}, end{t <= 100};

init idle;

trans idle -> sample_1{sync gop0!; assign c := 0;},
sample_1 -> pressure_2{guard c == 10; sync releasep0!; assign
hasrun := 1;},
pressure_2 -> pressure_3{sync gon0!; assign c := 0;},
pressure_3 -> controller_4{guard c == 10; sync releasen0!;},
controller_4 -> controller_5{sync gop1!; assign c := 0;},
controller_5 -> valve_6{guard c == 10; sync releasep1!;},
valve_6 -> valve_7{sync gon0!; assign c := 0;},
valve_7 -> actuator_8{guard c == 10; sync releasen0!;},
actuator_8 -> actuator_9{sync gop0!; assign c := 0;},
actuator_9 -> end{guard c == 10; sync releasep0!;},
end -> idle{guard t == 100; assign t := 0;};
}
```

Figure 6.10: The Textual Format of the `Control` Job

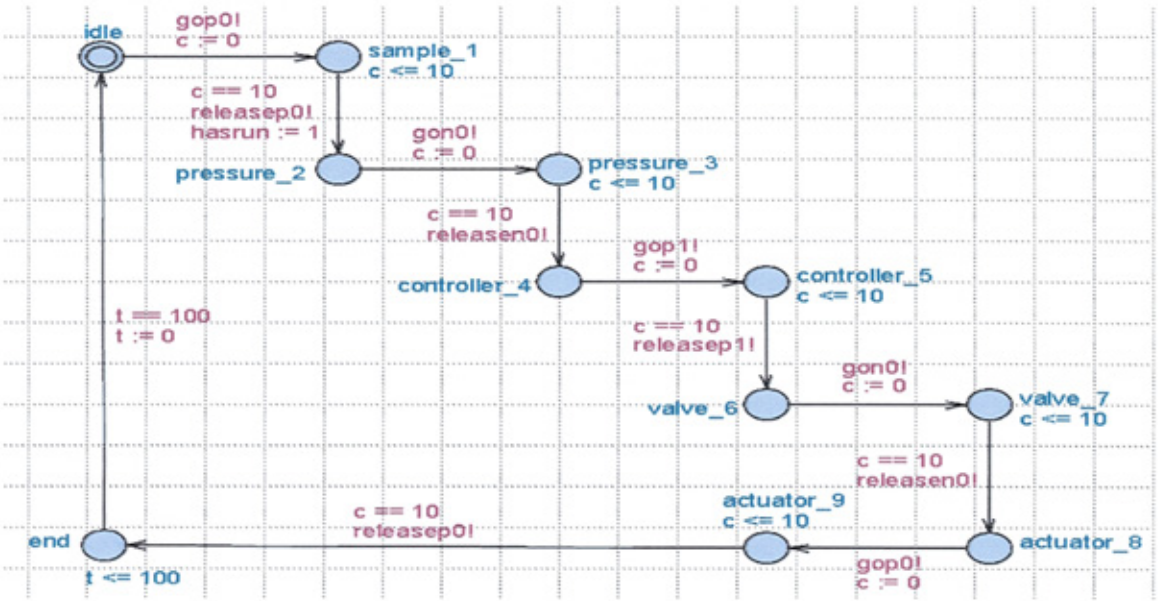


Figure 6.11: The Appearance of the `Control` Job in UPPAAL tool

Figure 6.11 shows the appearance of the textual format for the job in UPPAAL tool. However, the actual appearance is different from the Figure 6.11 when opening the job in the tool as all the positions of the states are automatically generated by the tool. However, it does not affect the operation of `Control` job; the only need is to make it pretty by moving the states to appropriate positions manually.

The Generator produces not only the textual format for jobs but also the textual format to control the accesses of each processor and network, called a scheduler. The scheduler is designed to allow its use by only one job at the same time; the textual formats for all schedulers are almost the same, but the difference is simply the synchronisation variables used in each scheduler, which they are already defined and allocated to processors, networks and jobs for their synchronisation. Figure 6.12 shows the scheduler for the `Plant` processor. This scheduler comprises only two states and two transitions and also has `gop0` and `releasep0` synchronisation variables which used to control its access.

```
Process Sche_Plant
{
state idle, S0;
init idle;
trans idle -> S0{sync gop0?;},
S0 -> idle{sync releasep0?;};
}
```

Figure 6.12: The Textual Format of the Plant Scheduler

With the UPPAAL models generated by the Generator according to the system specification in XML, the Generator also creates a temporal logic property which represents a requirement to verify whether or not the models are satisfied via the UPPAAL verifier in order to generate schedules for systems. In a case of a satisfaction between the models and the property, the toolset will force the verifier to create a trace file for the satisfaction. The temporal logic property is simple. With `E<>` UPPAAL property, meaning that eventually there exists a path satisfying from an initial location, the requirement property includes that all the models for jobs can reach to the `idle` location, even going through all the locations in their models once within a given time. It is known that the given time is retrieved from the least common multiple of all the

jobs' period. Figure 6.13 presents the property for the FC system which is working with the `Control` and `Alarm` job, named `Rcontrol_job`, `Ralarm_job` respectively, within 100 time-units as known the least common multiple for all the jobs; `Ralarm_job.hasrun == 1` is used to check all the models for the jobs at least execute once.

```
E<>( Rcontrol_job.idle and Ralarm_job.idle and Ralarm_job.hasrun == 1
and time <= 100)
```

Figure 6.13: The Temporal Logic Property for the FC System

6.2.3.2 Trace Analyser

With the UPPAAL models and the property generated by the Timed Automata Generator saved separately into two files, the UPPAAL verifier is executed to verify them and then examine the results of the verification. The command to call the verifier is 'verifyta [options] UPPAAL_models_file property_file'. Among the options, the '-t' option is particularly employed, meaning that a trace file has to be created after the successful verification. Other options are also useful but not related to the toolset's purpose. The file will contain all states, transitions, delays and variables used in the models until the end of a successful verification.

The Trace Analyser aims to analyse the trace, subsequently to gain the required information to produce a schedule for a system representing as UPPAAL models. According to the trace from the verification for the FC system (See Figure 6.14), it includes basically three indicators: `State:`, `Transitions:` and `Delay:` indicating the current status (or information) of states, transitions and delays respectively. The beginning of the trace indicates that all the models are located in their `idle` states and all the variables such as clocks are allocated 0 value using `State:` indicator. The first transition with `Transitions:` indicator appears between the `Rcontrol_job` and the `Sche_plan` models via the synchronisation variable `gop0`. The transition indicates that the `idle` state in `Rcontrol_job` goes to the `sample_1` state and simultaneously the `idle` in `Sche_plant` goes to the `S0` state. After this transition, the trace indicates again all the states, that is, the only state of the `Sche_plant` is the `S0` and other states are still

in the same state, `idle`, as well as all the variables are still holding the same value 0. A timing delay is retrieved through the `Delay:` indicator which denotes an amount of time all the clocks in the models have reached. Such as `Delay:` 10 in the Figure 6.14 notices 10 time-units as timing delay; this means that all the clocks in the models are passed by 10 time-units. Based on these information retrieved from the trace, it is possible to analyse how much time which a job takes which a processor; eventually retrieves all the information for a system schedule. In addition, the size of a trace will be different depending on verification.

```
State:
( Sche_ttp.idle Sche_plant.idle Sche_consol.idle Rcontrol_job.idle
Ralarm_job.idle )
time=0 Rcontrol_job.c=0 Rcontrol_job.t=0 Ralarm_job.c=0 Ralarm_job.t=0
Rcontrol_job.hasrun=0 Ralarm_job.hasrun=0

Transitions:
  Sche_plant.idle->Sche_plant.S0 { true, gop0? }
  Rcontrol_job.idle->Rcontrol_job.sample_1 { true, gop0!,
Rcontrol_job.c:=0 }

State:
( Sche_ttp.idle Sche_plant.S0 Sche_consol.idle Rcontrol_job.sample_1
Ralarm_job.idle )
time=0 Rcontrol_job.c=0 Rcontrol_job.t=0 Ralarm_job.c=0 Ralarm_job.t=0
Rcontrol_job.hasrun=0 Ralarm_job.hasrun=0

Delay: 10

State:
( Sche_ttp.idle Sche_plant.S0 Sche_consol.idle Rcontrol_job.sample_1
Ralarm_job.idle )
time=10 Rcontrol_job.c=10 Rcontrol_job.t=10 Ralarm_job.c=10
...
```

Figure 6.14: A Part of the Trace File for the FC System

6.2.3.3 Graphic Viewer

The output of the Trace Analyser is used by Graphic Viewer in order to produce a visual representation of the generated schedule. Figure 6.15 shows the graph for the schedule of the FC system. It is noted that the rows are labelled with the processor names or network name, and the columns indicate time values. The colours in the graph represent the jobs for the FC system: the Blue colour represents the schedule of the `Control` job; the Yellow denotes the schedule for the `Alarm` job. Additionally each square on the

graph represents a task or a message, whose name and computation or transfer time are described on it.

According to the graph, the `Control` job having 100 time-units period starts with the `Sample` task at 0, followed by the `Pressure` message at 10, `Controller` task at 20, `Valve` message at 30; the `Actuator` task starts at 40 and finishes at 50. This schedule certainly satisfies the deadline of this job which has to complete all the tasks and messages within 100 time-units; it will execute at 100 over again. Moving now to the `Alarm` job, it starts with the `Alarm` task a little late at 10 as waiting a resource of the `Plant` processor currently occupied by the `Sample` task from the `Control` job, then succeeds to the `Alarm` message at 20, `indicator` task at 30. The `Alarm` job can complete within 50 time-units as its period. However, there is a second schedule for it within 100 time-units because of its period. This time, the job does not have any disturbance from the `Control` job and so the `Alarm` task, `Alarm` message and `indicator` task start at 50, 60, and 70 respectively. Still, this schedule can also satisfy the second `Alarm` job.

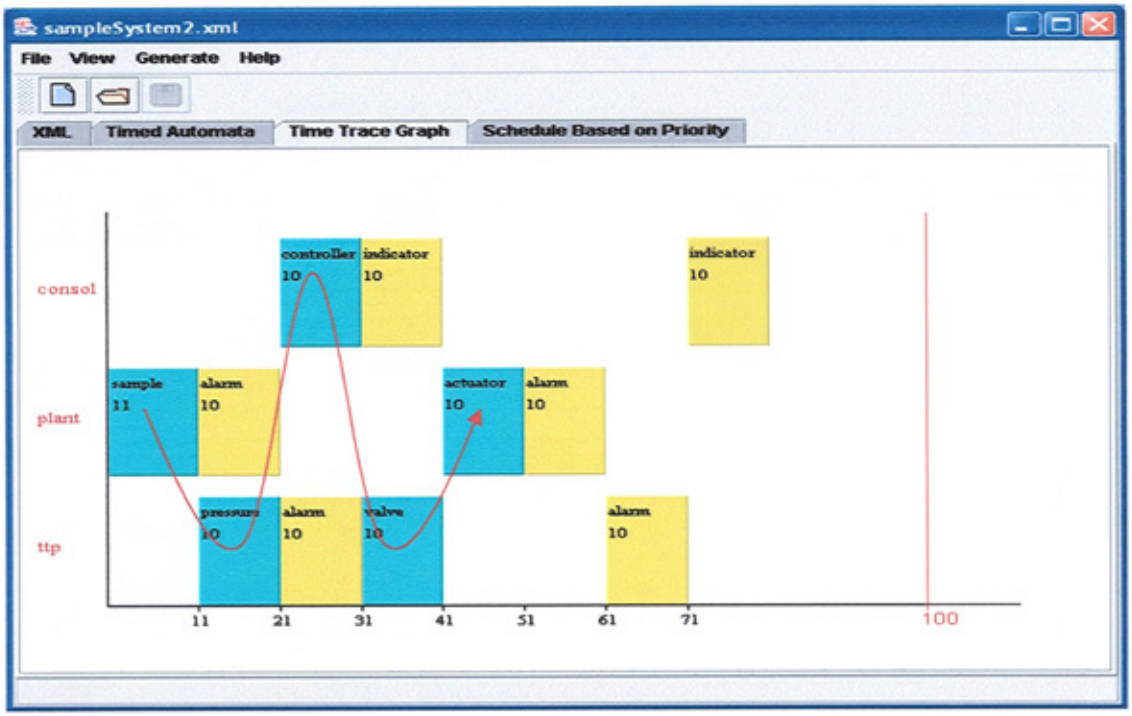


Figure 6.15: The Generated Schedule for the FC System

6.2.3.4 Schedule Extractor

Through the output of the Trace Analyser, it is possible not only to produce a visual graph but also to extract schedules for each processor and network using Schedule Extractor. According to the schedule gained by analysing the FC system by the toolset, it is understood that the `Plant` processor has four tasks allocated within 100 time-units. Two tasks are from the `Control` job and other two are from the `Alarm` job. First of all, the `Sample` task from the `Control` job starts at 0 and occupies the resource of the `Plant` processor at first; then the `Alarm` task in the `Alarm` job takes over the resource at 10. Between 20 and 40 in the processor there is not occupation from any jobs. The processor is occupied again at 40 by the `Actuator` task in the `Control` job and at 50 by the `Alarm` task in the `Alarm` job. The processor is free after 60.

For the `Control` processor, the `Controller` task in `Control` job at first starts at 20 as waiting a message from the precedent, `Sample` task. And then the `Indicator` task takes the `Consol` processor at 30 and 70 because the `Alarm` job having 50 time-units for its period works two times within 100 time-units. In the `Ttp` network, the `Pressure` and `Valve` messages from the tasks in the `Control` job are passed at 10 and 30 respectively. The same message, `Alarm` message, from the `Alarm` job starts at 20 and 60. As a result, there is the entire table of each processor and network for the FC system shown in Table 6.2. The Schedule Extractor provides this schedule as a textual format, thus it is possible to readily apply the schedule to any other application.

Plant Processor			Consol Processor			Ttp Network		
Start Time	End Time	Task	Start Time	End Time	Task	Start Time	End Time	Message
0	10	Sample	20	30	Controller	10	20	Pressure
10	20	Alarm	30	40	Indicator	20	30	Alarm
40	50	Actuator	70	80	Indicator	30	40	Valve
50	60	Alarm				60	70	Alarm

Table 6.2: The Schedule Table for the FC System

6.3 Comparison of the Proposed Approach

There are a number of algorithms in the literature to find a static schedule for distributed real-time systems. Many of these are heuristic approach and thus cannot guarantee to find a schedule even if one exists. As an example, a simple heuristic approach based on a heuristic algorithm by Burns [BHR95] is considered here. It is based on the premise that a job with the shortest period will be considered for allocation at first. This approach starts with an empty schedule and then inserts a job with the shortest period into the empty schedule at a time.

Consider the following system, `System 1`, shown in Figure 6.16 to illustrate the construction of a feasible schedule using the heuristic approach. The system consists of two processors (`Processor 1` and `Processor 2`) and one network (`Network 1`). There are 3 jobs in the system, named `Job A`, `Job C` and `Job C`. These jobs have similar behaviours with two tasks and one message. It is assumed that the computation times and transfer times of tasks and messages in `Job A` are 15 time-units; 10 time-units are assigned to the tasks and message in `Job B` and `Job C` has 20 time-units for both. The periods of the `Job A` and `Job B` are 50 time-units and the period of the `Job C` is 100 time-units. These assigned timing values for the system are listed in Table 6.3.

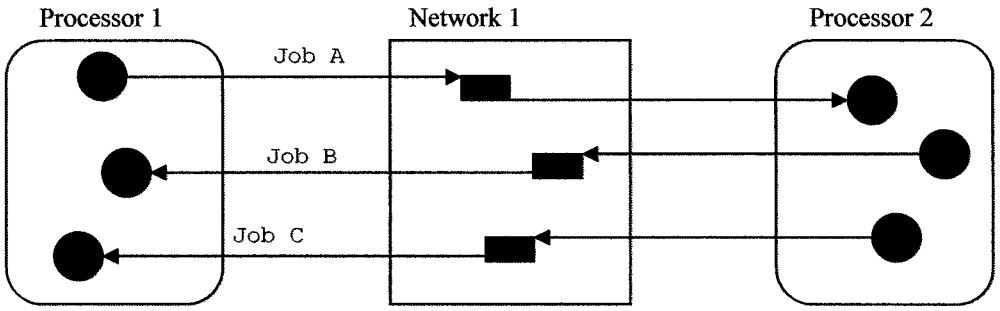


Figure 6.16: System 1

	Computation Time	Transfer Time	Period
Job A	15	15	50
Job B	10	10	50
Job C	20	20	100

Table 6.3: The Detail of the Jobs in System 1

According to the heuristic approach, it is expected that the Job A and Job B, which have 50 time-units for their period, will be allocated first and then the Job C, which has 100 time-units for its period, will be considered. However, this approach has a problem when allocating the Job C in schedule space (see Figure 6.17). After the Job A and B are allocated first, the first task of the Job C only starts at 30 time-units and thus this does not provide enough space for the following message and task of the job. In this case, this heuristic approach fails to find a feasible schedule.

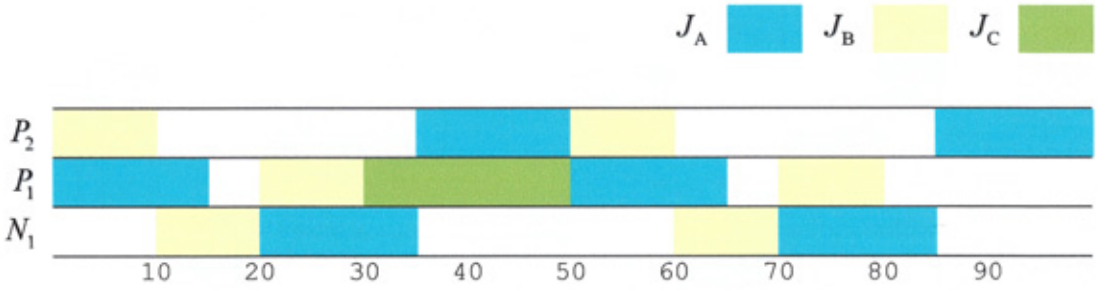


Figure 6.17: the Schedule Problem for System 1 based on the Heuristic Approach

In fact, this scheduling problem is simply solved by exchanging the beginning task ① of the Job C with the first finishing task ② of the Job B, as shown in Figure 6.18, In this case, the Job C can start at 15 time-units and thus the following message and task can be allocated at 35 time-units and 60 time-units respectively as having enough scheduling space for the task and message.

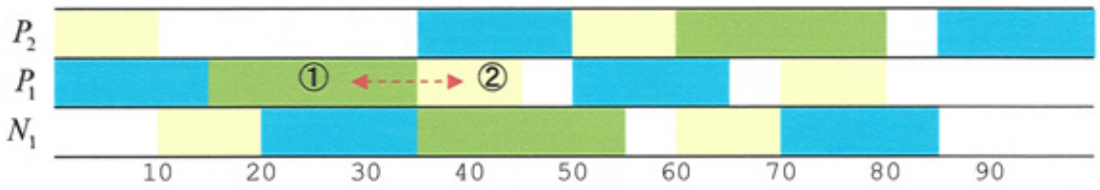


Figure 6.18: The Schedule Solution for System 1 based on the Heuristic Approach

For such a solution, it may be possible to consider other functions in this heuristic approach which exchange scheduling positions between tasks or between messages like a neighbour function in simulated annealing or a crossover function in genetic algorithm. However, these functions select two tasks at random in order to exchange the

positions of the tasks in the schedule. The algorithms do not necessarily exchange the task ① of the Job C and the task ② of the Job B and may not find feasible schedule. There are also local minima problems in the functions which may reduce opportunities to swap the positions between tasks and between messages. Consequently, it still will not guarantee to find a feasible schedule even with these functions.

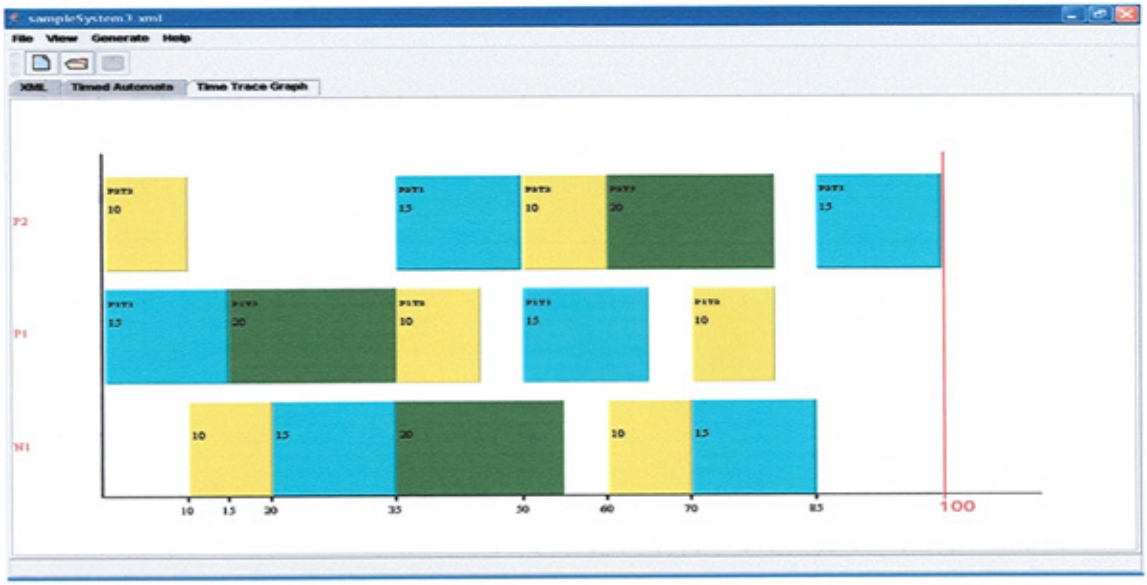


Figure 6.19: The Schedule using the Proposed Approach

The model checking approach proposed here can solve this problem easily as it examines all scheduling behaviours of the system by performing an exhaustive state space search. Compared with heuristic and other approaches, this proposed approach does not include intuition, experience and any function that randomly selects tasks and messages, and it even does not have local minima problems. Thus, it guarantees to find a feasible schedule if one exists and it will fail to find any schedule if none exists. Figure 6.19 shows the schedule of the system using the proposed approach which is the same result as Figure 6.18. As explained earlier, timed automata models and temporal logic properties representing the behaviours and requirements of the system are only required in the proposed approach. A feasible schedule is automatically computed by formally verifying that the property is satisfied with the given models. However, there

remains an issue of the state explosion problem particularly when considering complex and large systems and this will affect time to find a feasible schedule.

6.4 Summary

The proposed approach is that if UPPAAL models representing a system are satisfied in the given property, the UPPAAL verifier can force to produce a trace file as a successful example for its verification. Then, the trace is used to generate a schedule for the system. However, there are a couple of practical problems in the approach. First, there is a difficulty of creating UPPAAL models as this requires considerable time to properly create them with the fully understanding of UPPAAL tool and even timed automata theory. Second, the trace generated by the verifier is complicated to analyse as it includes many transition and state information for the verification. Finally, it is required to show the schedule in user-friendly manner such as by a graphical view rather than just a textual format. For these reasons, the prototype toolset is introduced.

The tool has mainly two phases: Preprocessor and Analyser Engine. In the Preprocessor, it is possible to describe the behaviours and constraints of a desired system with XML. Based on the terms defined previously such as System, Processor, Task, etc, each of these terms is defined with XML. The Analyser Engine phase includes Timed Automata Generator, Trace Analyser, Graphic Viewer and Scheduler Extractor. The Generator generates the textual format of UPPAAL models for jobs, processors and networks according to the given system specification expressed in XML. Thus the generated models are capable of working in the UPPAAL tool. The Generator also creates a temporal logic property to verify whether or not the models are satisfied via the UPPAAL verifier. With the verification outcome of the generated models and property as produced a diagnostic trace file, the Trace Analyser is to analyse the trace, consequently to gain the own required information to produce schedules for a system representing as the UPPAAL models. The Graphic Viewer and Scheduler Extractor are used to produce a visual graph and to extract schedules for the given system respectively.

A simple heuristic approach has been introduced to demonstrate many existing approaches cannot guarantee to find a schedule even if one exists. With the system

shown in the Figure 6.16, this heuristic approach has failed to find a schedule. However, the proposed approach based on model checking has shown a feasible schedule easily as examining all scheduling behaviours of the system.

Chapter 7

Case Studies

It is necessary to demonstrate the applicability and usability of the proposed technique of generating time-triggered schedules. Two case studies are included in this chapter, an Adaptive Cruise Control system and a Robot Transport system; these are typically found in automobile and manufacture industry respectively. These systems comprise a number of processors and networks, and particularly have activities highly requiring stringent timing constraints. Thus, the systems require deterministic schedules not only for each processor but also for over all processors and networks, in order to guarantee the participated tasks in the activities meet their deadlines. Assuming that the systems are based on time-triggered architecture, the first case study with the Adaptive Cruise Control system has been used to illustrate the way of how to apply the system into the approach using the toolset; the second case study with the Robot Transport system has been chosen to explore the approach with more tasks, messages and jobs, and in particular, more variable in its component behaviours.

7.1 Case Study: Adaptive Cruise Control System

7.1.1 Description

Automobile systems are increasingly being controlled by electronic devices over recent decades. Such automobile control technologies using the devices are expected to yield many benefits such as saving fuel uses, improving driving safety, reducing manufacture costs, etc. One of the technologies already fitted to modern vehicles is a Cruise Control system which helps to keep the constant speed of a vehicle without continuous pressure on an accelerator pedal and thus assists a driver such as reducing the driver's tiredness

on long car journeys [Axe98]. Furthermore, the efforts on a cruise control system, these days, have been applied to an Adaptive Cruise Control system (called ACC system shortly below) such as in X-By-Wire architecture for safety-critical distributed real-time application [WNS+05]. The system is an enhancement of conventional cruise control systems which allow the ACC vehicle (equipped with the ACC system) to adapt the vehicle's speed to traffic environments [ACC05, FES+98]. The ACC vehicle has additional radar in front of the vehicle to detect whether any vehicle slows down in the vehicle's path within a certain safe distance. If a slowing vehicle is detected, the ACC vehicle will automatically slow down and then maintain the safe distance from the slowing vehicle. However, if any slowing vehicle does not exist in the distance, the ACC vehicle will accelerate to its set cruise control speed. Thus, the ACC system allows not only keeping automatically the set speed but also autonomously avoiding any collision without giving interventions from drivers.

7.1.2 Structure

The physical structure of the ACC system is found in the literature [ACC05, ÅMH+05, AV98, Gmb03, Gur05, SPB+00]. The ACC system consists of interconnecting operational control units via a communication network; there are typically four units involved in the system: Console Control Unit, Brake Control Unit, Adaptive Cruise Unit and Engine Control Unit. The controls' names used here may be slightly different from some appearing in the literature but generally are acceptable as having the same principle functionality. Figure 7.1 shows the layout of the structure and the functional description of the control units are as follows:

- The `Adaptive Cruise Unit` (ACU) has a radar sensor to detect whether any vehicle is present or not within a certain safe distance. With the consequent information from the sensor, this unit sends a request to the Engine Control Unit and the Brake Control Unit. Thus it can manage the set speed and also maintain the clearance between the ACC vehicle and any vehicle detected in the safe distance.
- The `Engine Control Unit` (ECU) connecting with a vehicle's engine receives information from the ACU such as the speed set by a driver or the speed

controlled by the ACU when any slowing vehicle is detected within the safe distance. Then the ECU controls the vehicle's speed based on the information.

- The `Brake Control Unit` (BCU) has brake actuators and speed sensors in each wheel to determine the vehicle speed. The BCU is controlled by information sent from the ACU. Whenever it is necessary to decelerate the vehicle, the BCU will apply the brake actuators and then send the current vehicle speed to the ACU and ECU.
- The `Console Control Unit` (CCU) processes the cruise switches and sends its information to the ACU. The CCU also displays the current telltale information regarding the state of the ACC system received from the ACU.

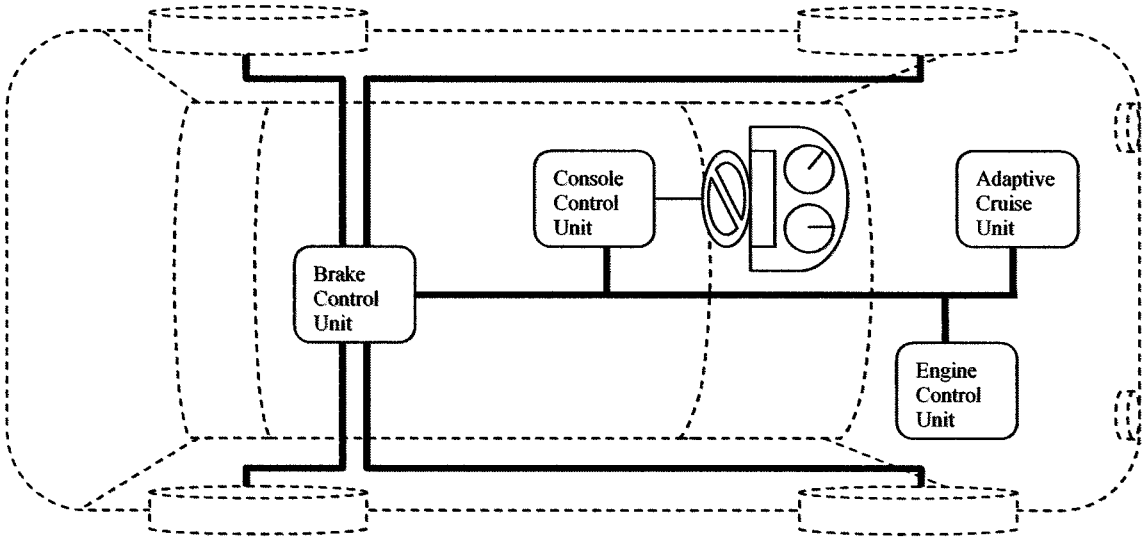


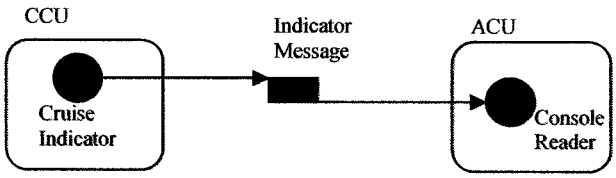
Figure 7.1: The Layout of Physical Structure of the ACC System

7.1.3 Operation

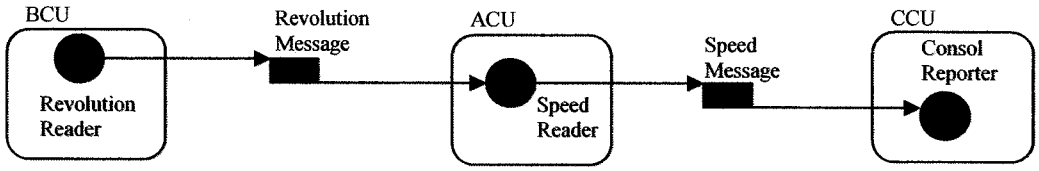
Although automobile systems including the ACC system are mostly based on event-triggered architectures, there are increasing demands to apply time-triggered architectures in future vehicles such as brake-by-wire, steer-by-wire, collision detection system, etc. The key design factor in time-triggered architectures is the requirement of all the information about tasks and messages priori in a system, in term of creating a pre-determined schedule. According to the structure of the ACC system and the description of each control unit, all the tasks and messages in the ACC system are

defined periodically and four operational jobs, thus, are assumed in the system as follows:

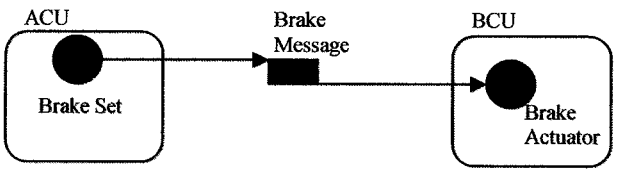
- `Console Cruise Job` is to cope with any cruise setting from a console unit such as switching on or off, resuming set speed, increasing or decreasing the cruise set speed, changing clearance, etc. This job sends any setting information on the CCU to the ACU periodically. It consists of two tasks and one message as shown below: `Cruise Indicator` task in the CCU, `Console Reader` task in the ACU and `Indicator` message.



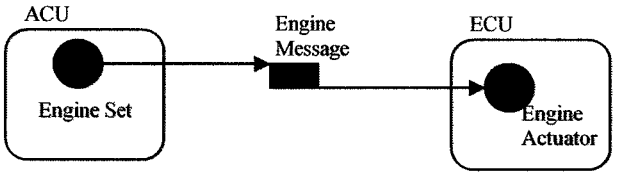
- `Console Report Job` is to display the current status of the ACC system on the console screen such as the current setting speed of the system. The job reads revolution data from wheels and then sends the data to the ACU. With considering the data and also other data such as from cruise radar attached on the ACU, the message for speed information will be sent to the CCU and then be displayed on the console screen. It consists of three tasks and two messages: `Revolution Reader` task in the BCU, `Speed Reader` task in the ACU and `Console Reporter` in the CCU, and `Revolution` and `Speed` message.



- `Brake Cruise Job` is to control speed, in particular, whenever the ACC system requires decelerating. This job works with two tasks and one message: `Brake Set` in the ACU, `Brake Actuator` in the BCU and `Brake` message.



- `Speed Control Job` is to control the current speed whenever the ACC system requires accelerating. The job involves two tasks and one message: `Engine Set` in the ACU, `Engine Actuator` in the ECU and `Engine` message.



With these jobs, the entire operational jobs for the ACC system are depicted in Figure 7.2.

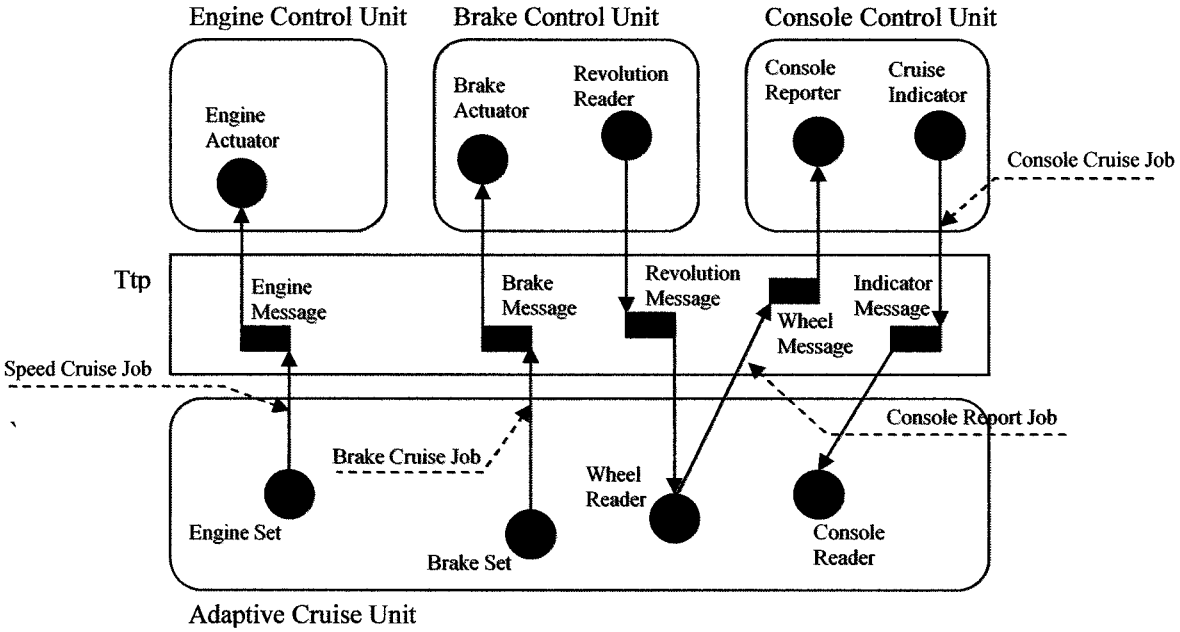


Figure 7.2: The Operational Jobs in the ACC System

It is assumed that the `Speed Cruise Job` and the `Brake Cruise Job` are occurred every 50 time-units as they are frequently required in the ACC system; the `Console Cruise Job` has 100 time-units period; the `Console Report Job` has 200 time-units period. Due to the difficulty in measuring the exact computation time for all the tasks and messages in the ACC system, the computation of all the tasks and the transferring time for all the message are assumed to be 10 time-units arbitrarily. Table 7.1 and Table 7.2 show the assumed times in the system.

Processor	Task	Computation	Period	PC_In	PC_Out
Adaptive Cruise Unit	Engine Set	10	50	N/A	Engine Message
	Brake Set	10	50	N/A	Brake Message
	Wheel Reader	10	200	Revolution Message	Wheel Message
	Console Reader	10	100	Indicator Message	N/A
Brake Control Unit	Brake Actuator	10	50	Brake Message	N/A
	Revolution Reader	10	200	N/A	Revolution Message
Engine Control Unit	Engine Actuator	10	50	Engine Message	N/A
Console Control Unit	Console Reporter	10	200	Wheel Message	N/A
	Cruise Indicator	10	100	N/A	Indicator Message

Table 7.1: Allocated Timing Values for the Processors in the ACC System

Network	Message	Transfer	Period	PC_In	PC_Out
Ttp	Engine Message	10	50	Engine Set	Engine Actuator
	Brake Message	10	50	Brake Set	Brake Actuator
	Revolution Message	10	200	Revolution Reader	Wheel Reader
	Wheel Message	10	200	Wheel Reader	Console Reporter
	Indicator Message	10	100	Cruise Indicator	Console Reader

Table 7.2: Allocated Timing Values for the Network in the ACC System

7.1.4 System Description with XML

In order to apply the proposed approach to the ACC system using the prototype toolset, it is required to express all the tasks, messages and jobs with XML as described in previous chapter (see Chapter 6.2.2). However, due to the limitation of space the complete XML expression for the system (see Appendix B) is not given here but there are some extracts below. Considering the `Cruise Control Unit`, for example, there are two tasks in the unit, the `Cruise Indicator` and `Console Reporter` tasks. These tasks can be defined with their periods, computation times and precedence constraints information with XML. The `Cruise Indicator` task, according to Table 7.1, is defined with 100 time-units for its period, 10 time-units for the computation time, and `N/A` and `Indicator Message` for the precedence constraint; the `Cruise Reporter` is also

defined with 200 time-units for the period, 10 time-units for the computation time, and `Wheel Message` and `N/A` for the precedence constraint. The XML expression of the `Cruise Control Unit` is described in Figure 7.3.

```
...
<processor>
   <processorID> ConsoleControlUnit </processorID>
   <task>
      <taskID> CruiseIndicator </taskID>
      <period> 100 </period>
      <computation> 10 </computation>
      <pc_in> N/A </pc_in>
      <pc_out> IndicatorMessage </pc_out>
   </task>
   <task>
      <taskID> ConsoleReporter </taskID>
      <period> 200 </period>
      <computation> 10 </computation>
      <pc_in> WheelMessage </pc_in>
      <pc_out> N/A </pc_out>
   </task>
</processor>
...
```

Figure 7.3: Description of the Console Control Unit with XML

```
...
<network>
   <message>
      <messageID> WheelMessage </messageID>
      <period> 200 </period>
      <transfer_time> 10 </transfer_time>
      <pc_in> WheelReader </pc_in>
      <pc_out> ConsoleReporter </pc_out>
   </message>
   ...
</network>
...
```

Figure 7.4: Description of the Wheel Message with XML

In a case of `Wheel Message` which is employed to communicate between the `Wheel Reader` and the `Control Report` tasks, the message can be defined with the period, transfer time, and precedence constraint information with XML. The message has 200 time-units for the period and 10 time-units for transferring time and `WheelReader` and `ConsoleReporter` for its predecessor and successor respectively. Figure 7.4 shows the expression of the message with XML.

```
...
<job>
   <jobID> ConsoleCruiseJob </jobID>
   <from>
      <when> start </when>
      <taskID> CruiseIndicator </taskID>
      <processorID> ConsoleControlUnit </processorID>
   </from>
   <to>
      <when> end </when>
      <taskID> ConsoleReader </taskID>
      <processorID> AdaptiveCruiseUnit </processorID>
   </to>
   <within> 100 </within>
</job>
...
```

Figure 7.5: Expression of the Console Cruise Job with XML

In a case of the `Console Cruise Job` which starts with the `Cruise Indicator` task in the CCU and ends with `Console Reader` in the ACU, the job can be defined with these tasks information used to express where it starts and ends; it is known that the job has 100 time-units as its period. See its description with XML in Figure 7.5. The remainder expression for the ACC system is attached in Appendix B.

7.1.5 Outputs from the Prototype Toolset

When applying the prototype toolset to the ACC system described with XML, outputs such as timed automata models, a property for the models, a trace file, a visual graph and schedules for each processor and network are expected. The outputs are briefly discussed here. First, considering the output of the timed automata models (or templates), there are nine UPPAAL models generated by the toolset. The four automata models are for the processors in the system to handle their resources and one model is for the network; the remaining four models represent the behaviour of each job. Figure 7.6 introduces a part of the text format for the `Engine Control Unit` scheduler model named `Sche_EngineControlUnit` and for the `Console Cruise Job` model named `RConsoleCruiseJob`. It is noticed that these names are slightly changed from their original names in terms of providing distinction between the models. The text format basically includes the information of the transitions and states, and synchronisation

between the models. The other models in the system are expected to have almost a similar pattern in the text format. Figure 7.7 shows the appearance of the `Console Report Job` model when it is opened in UPPAAL tool. It looks untidy. However, the appearance does not have any influence on its operation.

```
...

process Sche_EngineControlUnit
{
state idle, S0;
init idle;

trans idle -> S0{sync gop0?;},
S0 -> idle{sync releasep0?;};
}

...

process RConsoleCruiseJob
{
clock c, t;
int [0, 1] hasrun;

state idle, CruiseIndicator_1{c <= 10}, IndicatorMessage_2,
IndicatorMessage_3{c <= 10}, ConsoleReader_4, ConsoleReader_5{c <=
10}, end{t <= 100};

init idle;
trans idle -> CruiseIndicator_1{sync gop1!; assign c := 0;},
CruiseIndicator_1 -> IndicatorMessage_2{guard c == 10; sync
releasep1!; assign hasrun := 1;},
IndicatorMessage_2 -> IndicatorMessage_3{sync gon0!; assign c := 0;},
IndicatorMessage_3 -> ConsoleReader_4{guard c == 10; sync
releasen0!;},
ConsoleReader_4 -> ConsoleReader_5{sync gop3!; assign c := 0;},
ConsoleReader_5 -> end{guard c == 10; sync releasep3!;},
end -> idle{guard t == 100; assign t := 0;};
}

...
```

Figure 7.6: Timed Automata Models for the ACC System

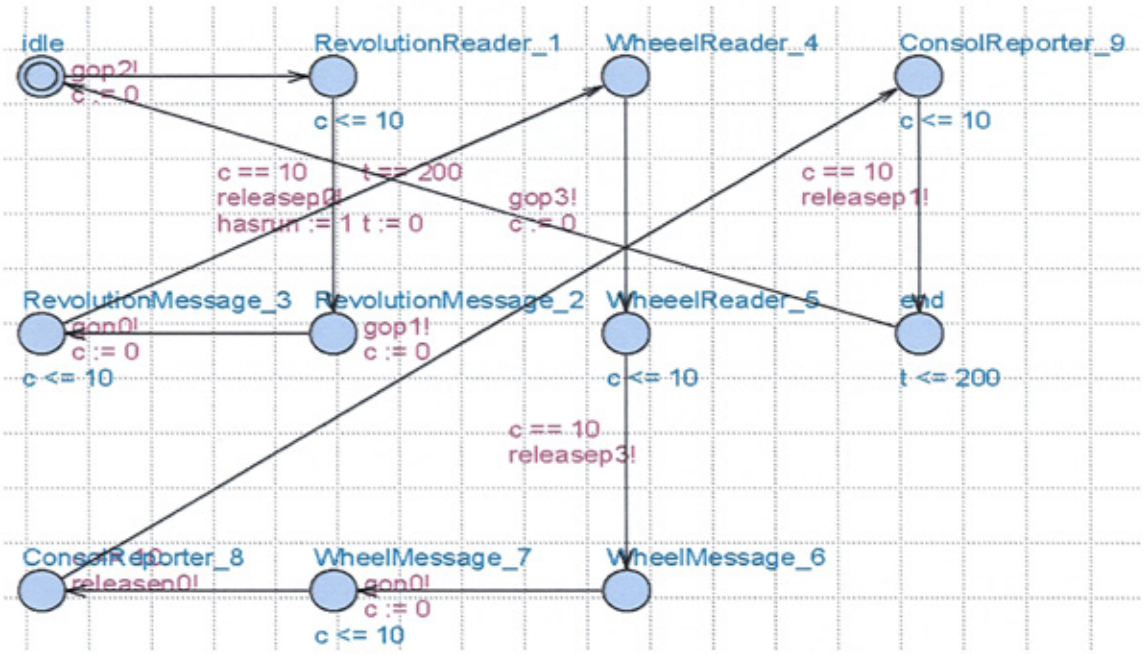


Figure 7.7: The Timed Automata Model for the Cruise Report Job

Second, as the `Console Cruise Job`, `Speed Cruse Job`, `Bake Cruise Job` and `Console Report Job` have the periods 100, 50, 50 and 200 time-units respectively, the scheduling round is expected 200 time-units as the least common multiple of all these jobs; this round is included in the property for verifying the models which is generated by the toolset. Thus, the property verifies that all the jobs can reach to their `idle` state within the scheduling round, 200 time-units and also all the jobs at least go through all the location in their model once (see Figure 7.8).

```
E<>( RConsoleCruiseJob.idle and RSpeedCruiseJob.idle and
     RBrakeCruiseJob.idle and RConsoleReportJob.idle and
     RConsoleReportJob.hasrun == 1 and time <= 200 )
```

Figure 7.8: The Property for the ACC System

Third, the approach generates a trace file using the generated models and property. Although the testing environment is limited - based on 512M memory and `Celeron CPU` 2.20GHz with Windows operating system, it does not take much time to generate the trace file in the environment. The size of the trace file is estimated around 67K

which includes all the transitions between states, and the delay information for all the clocks.

Fourth, the visual graph is produced by the tool after analysing the trace file and each colour on the graph represents one of the jobs in the ACC system. In the graph shown in Figure 7.9, a yellow colour represents the `Speed Cruise Job`; a green is for the `Brake Cruise Job`; a blue illustrates the `Console Cruise Job`; a tan is for the `Console Report Job`. According to the graph, it is noticed that the `Speed Cruise Job` starts at 0, 50, 100 and 150 time-units, the `Brake Cruise Job` at 10, 60, 110 and 160 time-units, the `Console Cruise Job` at 0 and 100 time-units, and the `Console Report Job` at 0. These timing values satisfy the periods of all the jobs in the ACC system and thus these can be used for a successful schedule for the system.

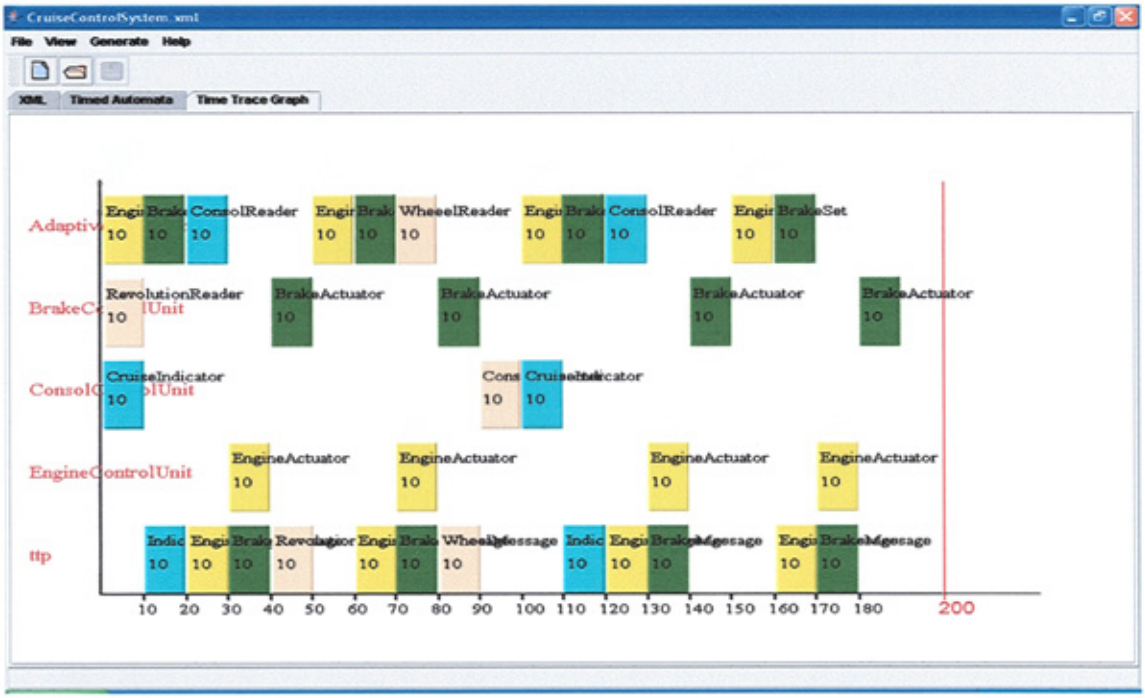


Figure 7.9: The Schedule for the ACC System

7.1.6 Schedule for the ACC System

Eventually, within 200 time-units each processor and network in the systems has the following schedules:

- The `Adaptive Cruise Unit` has 11 tasks which appear at 0 for the `Engine Set`, 10 for the `Brake Set`, 20 for the `Console Reader`, 50 for the `Engine Set`, 60

for the Brake Set, 70 for the Wheel Reader, 100 for the Engine Set, 110 for the Brake Set, 120 for the Console Reader, 150 for the Engine Set and 160 for the Brake Set. In particular, the Engine Set and Brake Set appear four times as their period are known 50 time-units. It is expected that the processor utilisation of this unit is approximately 55%.

- The Brake Control Unit has 5 tasks at 0 for the Revolution Reader, 40 for the Brake Actuator, 80 for the Brake Actuator, 140 for the Brake Actuator and 180 for the Brake Actuator. With these tasks, the processor utilisation of this unit is estimated about 25%.
- The Console Control Unit gets only 3 tasks at 0 for the Cruise Indicator, 90 for the Console Reporter and 100 for the Cruise Indicator. The processor utilisation is 15% which is even less than the Brake Control Unit.
- The Engine Control Unit has only the Engine Actuator at 30, 70, 130 and 170. The processor utilisation is approximately 20%.
- The Ttp network has 12 messages passing within 200 time-units. The messages are at 10 for the Indicator message, 20 for the Engine message, 30 for the Brake message, 40 for the Revolution Message, 60 for the Engine message, 70 for the Brake message, 80 for the Wheel message, 110 for the Indicator message, 120 for the Engine message, 130 for the Brake message, 160 for the Engine message and 170 for the Brake message. This network has the greatest utilisation than others units, which is known 60%.

7.2 Case Study: Robot Transport System

The previous case study of the ACC system has been focused on illustrating how to apply the approach using the prototype toolset, which requires extracting jobs from the system, describing the system with XML expression as an input to the toolset, analysing the timed automata models generated by the toolset, etc. This case study will explore more complicated system schedules with a Robot Transport system, which has almost three times as many jobs as the ACC system. In particular, timing influences of the approach will be examined and be more variable in the component behaviours of the system.

7.2.1 Description

Industrial plant developers have been trying to adapt an automatic manufacturing system for a long time as such systems are directly linked to the expense of labours and also the productivity of a company, etc. There are various kinds of such systems and a typical system employs robots for automatic manufacturing. Here, a Robot Transport system is introduced to move a product from one position to another position [IKL+00] – This system is simply a model of a real model system but includes most functionalities as the same as real ones, shown in Figure 7.10.

Figure 7.10: The Robot Transport System Model Picture

The scenario of the Robot Transport system is as follows: the system, in particular a control panel, is able to sense an item (a block in this model) whenever any item is left on a reserved loading position, then a load robot automatically picks the item up and drops it on the front of a conveyor – which is a drop place on the conveyor. However, if

there is already an item on the drop place, the load robot has to wait until the place is available. Whenever the conveyor has an item on the front of it, it transports the item to the back of it on which a unload robot can pick it up. The unload robot picks the item up as soon as possible and drops it on a reserved unloading position. Likewise, if any item is already occupied on the position, the unload robot has to wait until the position is available. It is expected that the two robots works simultaneously, meaning that many items are being loaded and unloaded at the same time.

7.2.2 Structure

As described above, the Robot Transport system consists of four control units: `Control Panel`, `Load Robot`, `Unload Robot` and `Conveyor`. These control units are interconnected via a network and communicated by message passing. There also are four sensors attached in the system to check the position of products; two are attached on the `Control Panel` and another two are attached on the `Conveyor`. Figure 7.11 shows the structure layout of the Robot Transport system and the functional description of each control unit is described as follows:

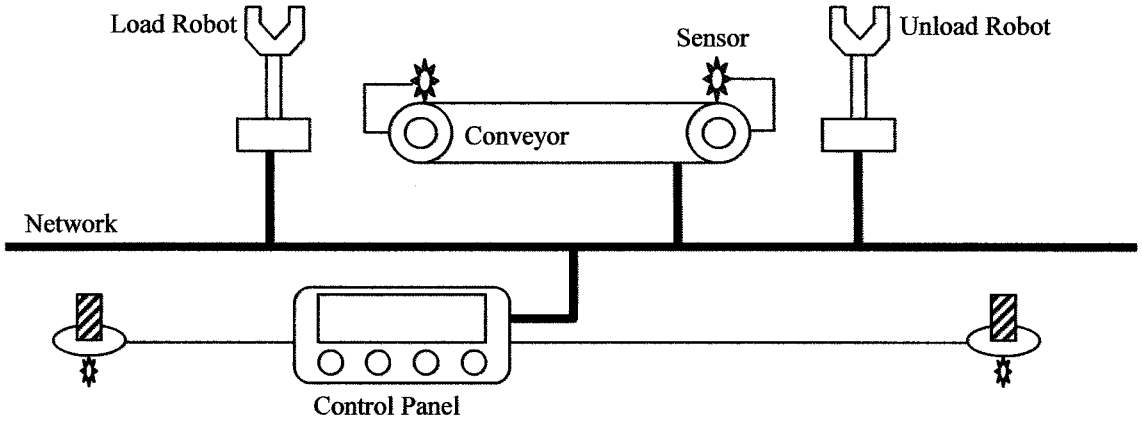


Figure 7.11: The Layout of Physical Structure of the Robot Transport System

- The `Control Panel` consists of a set of switches and push buttons which are used to control a system, and LEDs and a small screen to indicate and display the condition of a system. For example, the switches can be used to start or stop the system; the buttons can be used to signal emergency conditions, an item presence, etc; the red LED can be flashed in a case of system error and the green

LED is used during normal operations; any message is possible to be displayed on the small screen. There are also two sensors attached to the `Control Panel`. One of them is for checking whether or not any item is left on a reserved loading position and the other is for a reserved unloading position. The `Control Panel` has the responsibility of checking the switches, push buttons and sensors. Then, whenever one of them is activated, the `Control Panel` works with the `Load Robot` to pick the item up or works with the `Unload Robot` to drop it. The `Control Panel` also has the responsibility of displaying system conditions or messages passed by other control units using the LEDs and the small screen.

- The `Load Robot` has to pick an item up in the reserved loading position when it is asked from the `Control Panel`. Then, it drops the item on the front of the `Conveyor` if a drop place is available, in other words, there is not already any item on the front of the `Conveyor`. However, as the Robot Transport system allows many items available, it is expected that `Load Robot` requires communication with the `Control Panel` and the `Conveyor` periodically, in order to avoid any collision between items.
- The `Conveyor` comprises two sensors and a motor. As the two sensors are located near to each end, it is possible to detect whether an item has successfully been loaded and removed from the `Conveyor`. A motor is used to move the `Conveyor` belt to transfer an item from the front of the `Conveyor` to the back. Then, the transferred item will be removed by the `Unload Robot`. As this system is to allow as many items as possible to be on the `Conveyor`, it is expected that the `Conveyor` requires communication with the `Load Robot` and the `Unload Robot`.
- The `Unload Robot` has almost the same operational functionality as the `Load Robot`. It has a call from the `Conveyor` when any item reaches the back of the `Conveyor` and the robot picks the item up. Then, if any place is available in the reserved unloading position, the robot will drop it. As many items are available on the system, the Unload Robot needs a careful communication with the Conveyor and the Control Panel when picking or dropping items.

7.2.3 Operation

In order to define all the tasks and messages a priori which fulfil the requirements of the Robot Transport system in a time-triggered architecture, it is necessary to understand the system operations by the communication protocol between the control units based on the structure of the system and the description of each control unit. For example, when the `Load Robot` tries to move an item from a loading position to the `Conveyor`, what kind of overall messages are required to communicate between the `Load Robot` and the `Control Panel` and also between the `Load Robot` and the `Conveyor`. Although there may exist various communications for the system, generally 14 communication procedures are considered from picking an item up on the loading position till dropping the item on the unloading position. Figure 7.12 shows the overall communications and each procedure adopted:

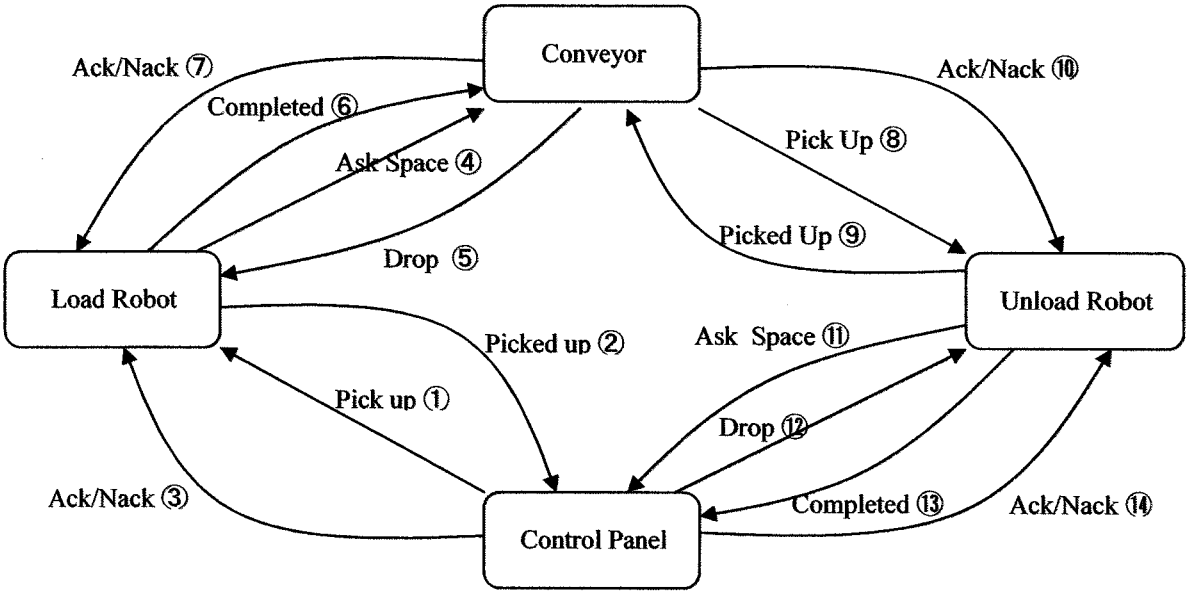


Figure 7.12: The Layout of Physical Structure of the Robot Transport System

①. When the `Control Panel` finds an item in the reserved loading position, it gives a command to the `Load Robot` to pick the item up.

②. After the `Load Robot` picks the item up, then it sends the completion of the picking to the `Control Panel`.

③. The `Control Panel` gives the `Load Robot` an acknowledgement whether the loading procedure is completed or not.

④. The `Load Robot` requests a drop space to the `Conveyor` after loading the item from the loading position.

⑤. If any space is available on the `Conveyor`, then the `Conveyor` gives a command to the `Load Robot` to drop the item on it.

⑥. The `Load Robot` drops the item on the `Conveyor` and then gives the completion of the dropping procedure.

⑦. The `Conveyor` gives an acknowledgement whether the dropping procedure is completed or not.

⑧. If the `Conveyor` senses an item on the back of the Conveyor, then it gives a command to the `Unload Robot` to pick the item up.

⑨. After the `Unload Robot` picks the item up, it sends a completion message.

⑩. The `Conveyor` gives an acknowledgement to the `Unload Robot` for the picking procedure.

⑪. The `Unload Robot` requests a drop space on the reserved unloading position to the `Control Panel`.

⑫. When any space is available on the unloading position, the `Control Panel` gives a command to the `Unload Robot` to drop the item on the position.

⑬. The `Unload Robot` drops the item and then gives the completion of the dropping procedure.

⑭. The `Control Panel` gives acknowledgement to the `Unload Robot` for the dropping procedure on the unloading position.

The overall communication protocol seems to be a sequence order from ① to ⑭ which means that there might be only one item available on the system. But, the `Load Robot` and `Unload Robot` work simultaneously with a different frequency depending on items available on the system and so it is possible to see many items working. However, considering the aspect that the robots do either picking or dropping an item in this communication protocol, it is expected that the loading procedure from the `Load Robot` represented by ①, ②, ③ will not be the same time as its dropping procedure represented

by ④, ⑤, ⑥, ⑦. Also, the procedure ⑧, ⑨, ⑩ between the `Unload Robot` and the `Conveyor` will not be the same time as the procedure ⑪, ⑫, ⑬, ⑭.

It seems that this communication protocol for the system operations may be well adopted as an event-triggered architecture rather than a time-triggered architecture. This is because the operations are discontinuous, depending on items available in the system and also some of the operations require an immediate response and some of the operations depends on an event (or result) from other operations such as when only the `Load Robot` picks up an item, then it can drop the item on the `Conveyor`, otherwise the `Load Robot` cannot proceed to the dropping operation. However, this case study is focusing on the design of scheduling on a time-triggered architecture and thus it is necessary to extract the operations (called jobs below) which can be adopted in a time-triggered architecture, based on the above communication protocol. Now, all the jobs are carefully defined with the tasks and messages which are occurred periodically, in terms of creating a pre-determined schedule for the system working with the time-triggered architecture. Eventually it is expected that there are 10 operational jobs in the Robot Transport system as shown in Figure 7.13. The direction arrow in the Figure will help to understand the progressing flow of the jobs; each is described below:

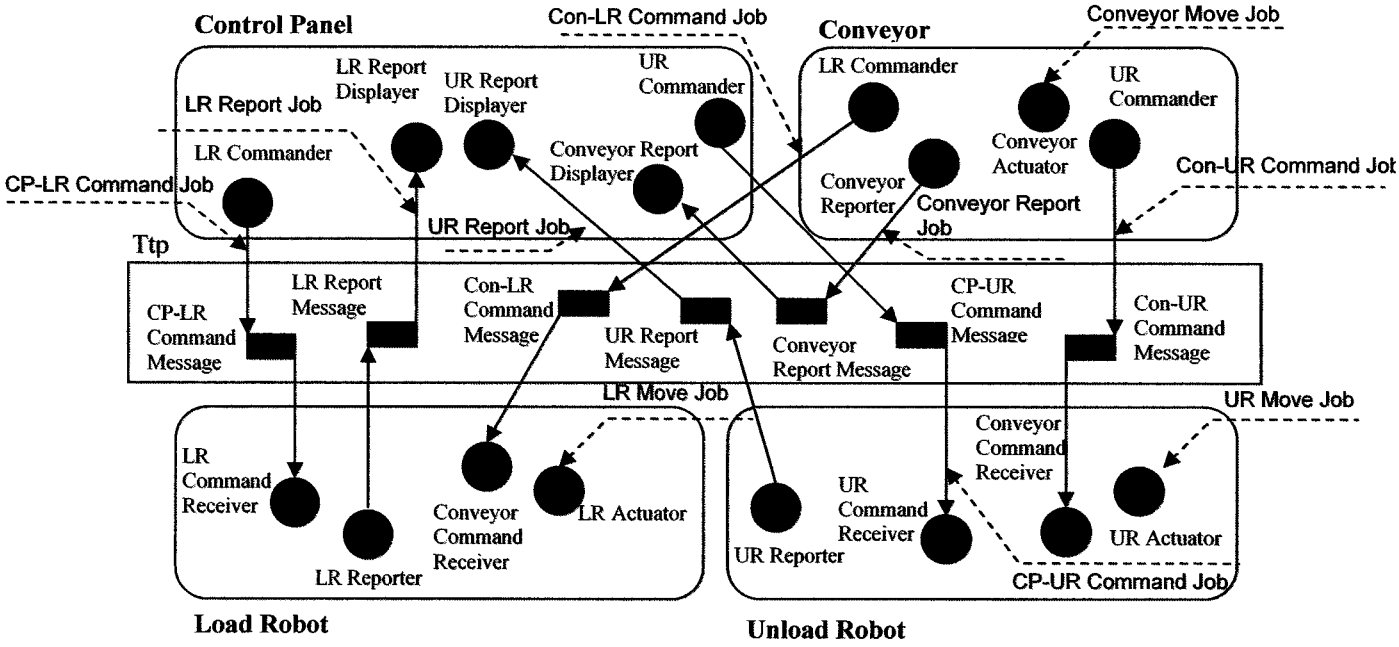


Figure 7.13: The Operational Jobs in the Robot Transport System

- The CP-LR Command Job sends a periodic command from the Control Panel to the Load Robot by message passing in order to respond to any change from the Control Panel such as switch on or off, button pressed, sensing an item on a loading position, etc; then the Load Robot obeys the command to perform its movement. This job consists of two tasks and one message: LR Commander in the Control Panel, LR Command Receiver in the Load Robot and CP-LR Command message.
- The LR Report Job periodically displays the current status of the Load Robot on the Control Panel. This job starts by reading the status of the Load Robot such as its movement and position, etc, and then sends a message with the status information. After receiving the message, the Control Panel displays the status using the LEDs or the small screen. It consists of the LR Reporter and LR Report Displayer tasks and the LR Report message.
- The LR Move Job moves the Load Robot depending on the commands passed from the Control Panel and the Conveyor. In a case of receiving the command from the Control Panel, the Load Robot will pick up an item. In a case of receiving the command from the Conveyor, the robot will drop an item on the Conveyor. The LR Actuator job comprises a single task and has no associated message.
- The Con-LR Command Job periodically drops an item from the Conveyor to the Load Robot whenever any space is available on the front of the Conveyor; otherwise the command will be ignored by the Load Robot. This job associates with LR Commander in the Conveyor and Conveyor Command Receiver in the Load Robot, and Con-LR Command message.
- The Conveyor Report Job communicates the current status of the Conveyor to the Control Panel and then displays it on the panel. This job also uses the LEDs or the small screen to indicate or displays the status information passed from the Conveyor. This job consists of two tasks and one message which are Conveyor Reporter and Conveyor Report Displayer, and the Conveyor Report message

- The Con-UR Command Job regularly sends a command to Unload Robot, in order to ask for picking up an item which is moved to the end of the Conveyor. Depending on the command passed by a message, the Unload Robot performs its movement. UR Commander in the Conveyor and Conveyor Command Receiver tasks; Con-UR Command message belong to this job.
- The Conveyor Move Job shifts the Conveyor belt. So, an item dropped on the front of the Conveyor can be shifted until the item reaches the end, recognised by a sensor. If the item on the end is picked up by the Unload Robot, the conveyor continuously shifts another item to the end. In this system, it is assumed that the shift direction is one-way. In this job, Conveyor Actuator task works for itself.
- The CP-UR Command Job periodically sends a command from the Control Panel to the Unload Robot in order to drop an item on the unloading position. The Unload Robot drops the item on the unloading position. If there is no space on the position, the command will be ignored by the robot. This job consists of two tasks and one message: UR Commander in the Control Panel, UR Command Receiver in the Unload Robot and CP-UR Command Message.
- The UR Report Job displays the current status of the Unload Robot on the Control Panel using the LEDs or the small screen. This job will help to monitor the Unload Robot such as its movement, positions, errors, etc. UR Reporter in the Unload Robot and UK Report Displayer in the Control Panel and UR Report message belong to this job.
- The UR Move Job moves the Unload Robot depending on the commands passed from the Conveyor and the Control Panel. When working with the command from the Conveyor, the Unload Robot will pick up an item from the Conveyor. When working with the command from the Control Panel, the Unload Robot will drop the item on the unloading position. It has only one task, UR Actuator task.

The periods for the jobs are defined to perform their requirements in a time-triggered architecture and the periods are assigned harmonic values in order to construct a scheduling round as the least common multiple of all the jobs. It is assumed that the

jobs taking the responsibility of reporting the current status to the Control Panel, such as LR Report Job for the Load Robot, UR Report Job for the Unload Robot and Conveyor Report Job for the Conveyor, are given 100 time-units for their periods. The command jobs from the Control Panel such as CP-LR Command Job and CP-UR Command Job, have more frequency than the command jobs from the Conveyor such as Con-LR Command Job and Con-UR Command Job, and thus the periods are given 200 time-units and 400 time-units respectively. The jobs related to the movement of the control units such as LR Move Job, UR Report Job and Conveyor Move Job, have the highest frequency in the system as 50 time-units. Table 7.3 summarize these defined jobs with their tasks and messages and their periods.

Job	Period	Task ->	Message ->	Task
CP-LR Command Job	200	LR Commander	CP-LR Command	LR Command Receiver
LR Report Job	100	LR Reporter	LR Report	LR Report Displayer
Con-LR Command Job	400	LR Commander	Con-LR Command	Conveyor Command Receiver
CP-UR Command Job	200	UR Commander	CP-UR Command	UR Command Receiver
UR Report Job	100	UR Reporter	UR Report	UR Report Displayer
Con-UR Command Job	400	UR Commander	Con-UR Command	Conveyor Command Receiver
Conveyor Report Job	100	Conveyor Reporter	Conveyor Report	Conveyor Report Displayer
LR Move Job	50	LR Actuator		
UR Move Job	50	UR Actuator		
Conveyor Move Job	50	Conveyor Actuator		

Table 7.3: Tasks and Message Belonging to the Jobs on the System

7.2.4 Description with XML

An XML expression for the Robot Transfer system is required to analyses the system using the prototype toolset, containing the information of processors, networks, tasks, messages and jobs in XML. According to Figure 7.13, the system consists of four control units and one network as known Control Panel, Load Robot, Unload Robot and Conveyor for processors, and Ttp for a network. The characteristics of the jobs

such as theirs periods, starting tasks and ending tasks are easily retrieved from the Table 7.3.

Processor	Task	Period	PC-In	PC-Out
Control Panel	LR Commander	200	N/A	CP-LR Command Message
	LR Report Displayer	100	LR Report Message	N/A
	UR Report Displayer	100	UR Report Message	N/A
	UR Commander	200	N/A	CP-UR Command Message
	Conveyor Report Displayer	100	Conveyor Report Message	N/A
Conveyor	LR Commander	400	N/A	Con-LR Command Message
	Conveyor Actuator	50	N/A	N/A
	UR Commander	400	N/A	Con-UR Command Message
	Conveyor Reporter	100	N/A	Conveyor Report Message
Load Robot	LR Command Receiver	200	CP-LR Command Message	N/A
	LR Reporter	100	N/A	LR Report Message
	Conveyor Command Receiver	400	Con-LR Command Message	N/A
	LR Actuator	50	N/A	N/A
Unload Robot	UR Command Receiver	200	CP-UR Command Message	N/A
	UR Reporter	100	N/A	UR Report Message
	Conveyor Command Receiver	400	Con-UR Command Message	N/A
	UR Actuator	50	N/A	N/A
Network	**Message**	**Period**	**PC-In**	**PC-Out**
Ttp	CP-LR Command Message	200	LR Commander	LR Command Receiver
	LR Report Message	100	LR Reporter	LR Report Displayer
	Con-LR Command Message	400	LR Commander	Conveyor Command Receiver
	CP-UR Command Message	200	UR Commander	UR Command Receiver
	UR Report Message	100	UR Reporter	UR Report Displayer
	Con-UR Command Message	400	UR Commander	Conveyor Command Receiver
	Conveyor Report Message	100	Conveyor Reporter	Conveyor Report Displayer

Table 7.4: The Information of the Tasks and Messages in the Robot Transfer System

Further, the periods and precedence constraints information for each task and message are derived from the defined jobs. It is known that the period of each task and

message must have the same period of the job they are belonging to. Thus, the `LR Commander` task belonging to the `CP-LR Command Job`, for example, will have 200 time-units for its period which is the same as the period of `CP-LR Command Job`, 200. With reference of the direction of the arrows in the Figure 7.13, the precedence constraint of each task and message is retrieved. For example, the `LR Command Receiver` in the `Load Robot` has a predecessor but does not have a successor as it is the last task in the `CP-LR Command Job`. Table 7.4 summarises the tasks and messages for the system. The overall expression for the Robot Transport system is attached in Appendix C.

7.2.5 Experimental Studies

When generating schedules in the proposed approach, some factors which may affect the schedules are identified such as the time taking to generate schedules is different depending on the given times to tasks, messages and jobs. Here, four experimental studies have been performed to examine the scheduling of the Robot Transport system.

7.2.5.1 Influence of Arbitrary Time Units – Study A

Arbitrary times are allocated to the tasks, messages and jobs in the Robot Transport system as their periods, the computation times, transfer times, etc. However, these units can be redefined in cases that either a higher time unit or lower time unit is preferable. For example, the periods of the jobs already defined with 50, 100, 200 and 400 time-units can be redefined with 500, 1000, 2000, and 4000 time-units or even higher. Thus, it is worthwhile to investigate how an arbitrary time unit gives the influences generating schedules. In particular, it is interesting to investigate the time required to generate a schedule.

Unit: Time-Units

	Periods	**Computation and Transfer Time**
System A.1	50, 100, 200, 400	10
System A.2	5, 10, 20, 40	1
System A.3	500, 1000, 2000, 4000	100

Table 7.5: System Set A

Consider the following set of systems, `System Set A` (see Table 7.5), which are the same behaviour as the Robot Transport system but include different arbitrary time-units on the tasks, messages and jobs.

- `System A.1` - assuming that the periods of the jobs are defined with one of 50, 100, 200 and 400 time-units, and the computation times and transfer times for all the tasks and messages are assigned 10 time-units.
- `System A.2` - assuming that the periods of the jobs has one of 5, 10, 20 and 40 time-units, and all the tasks and messages have 1 time-unit. Generally, these assigned timing values are 10 times less than `System A.1`.
- `System A.3` - assuming that the periods of the jobs has one of 500, 1000, 2000 and 4000 time-units, and all the tasks and messages are given to 100 time-units. These timing values are generally 10 times greater than `System A.1`.

Unit: Seconds

	System A.1	System A.2	System A.3
Attempt 1	24	26	24
Attempt 2	25	24	25
Attempt 3	25	24	25
Attempt 4	24	24	24
Average	24.50	24.50	24.50

Table 7.6: Time Required to Find Schedules for System Set A

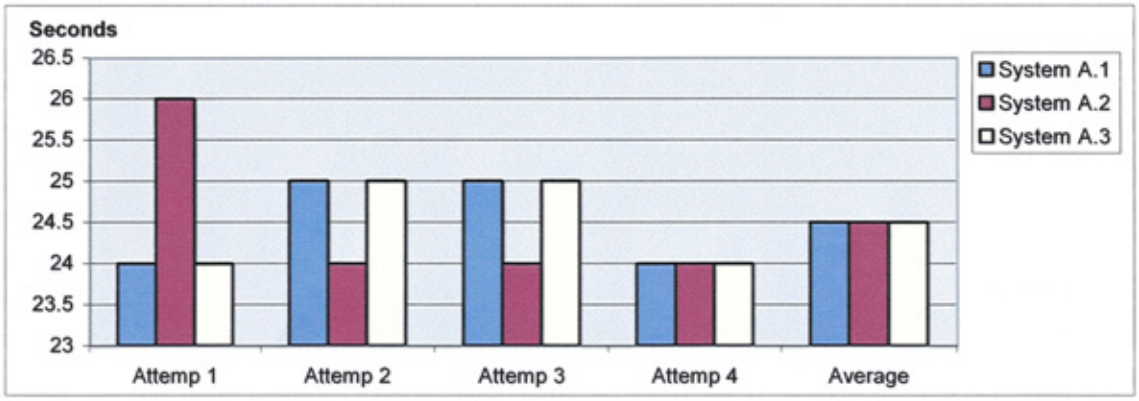


Figure 7.14: Time Graph for Table 7.6

The System Set A is analysed four times using the prototype toolset; the time consumed for each system was measured until the toolset produced a schedule. According to the timing result shown in Table 7.6, there is no significant time differences in the System Set A which are reaching around between 24 – 26 seconds, and 24.50 seconds in average. The small difference can be neglected as it is tiny and random, affected by other factors, i.e. it is difficult to give 100% pure CPU resource to this experimental study testing on Windows XP operating system because other processors are also executing on it. From the Study A, it is understood that the arbitrary times for periods, computation times and transfer times do not have an influence in the approach and consequently, any arbitrary units can be considered in the approach such as a second or millisecond, etc.

7.2.5.2 Influence of Appearance of Jobs – Study B

The schedules of the Robot Transport system are expected to be complete within 400 time-units; the value 400 time-units is called as the scheduling round for the system and is retrieved from the lease common multiple of all the jobs whose periods are one of 50, 100, 200 and 400 time-units. Within the round it is clearly known how many times a job may appear. For example, in the system the LR Move Job having 50 time-units as its period will be appeared 8 times within the round 400 time-units; the LR Report Job having the period 100 time-units will be 4 times; the Con-UR Command Job will be appeared once because its period is the same as the round. Accordingly, the number of appearances of a job depends on the length of its period; the shorter the period, the more frequently it appears in a scheduling round.

However, it is interesting to examine if the number of appearances of a job in a scheduling round affects the complexity of schedules; this may also affect the approach. For this reason, the Study B investigates by how much the number of appearances of a job affects the time required to generate schedules. With the jobs in the Robot Transport system but ignoring the periods already allocated above, the periods of all the jobs are newly assigned to 400 time-units and the computation times and transfer times to 10 time-units in the Study B. And then, the period of one of the jobs is decreased to 100 time-units in order to make it appear 4 times within 400 time-units and the time required

to generate schedules via the toolset is measured. This is continued gradually until only one of the jobs is left with its period 400 time-units – which can keep the scheduling round as 400 time-units.

Unit: Time-Units

Job \ System	B.1	B.2	B.3	B.4	B.5	B.6	B.7	B.8	B.9	B.10
LR Report Job	400	100	100	100	100	100	100	100	100	100
UR Report Job	400	400	100	100	100	100	100	100	100	100
Con-LR Command Job	400	400	400	100	100	100	100	100	100	100
Con-UR Command Job	400	400	400	400	100	100	100	100	100	100
LR Move Job	400	400	400	400	400	100	100	100	100	100
UR Move Job	400	400	400	400	400	400	100	100	100	100
Conveyor Move Job	400	400	400	400	400	400	400	100	100	100
CP-LR Command Job	400	400	400	400	400	400	400	400	100	100
CP-UR Command Job	400	400	400	400	400	400	400	400	400	100
Conveyor Report Job	400	400	400	400	400	400	400	400	400	400

Table 7.7: System Set B

Consider the following set of systems, `System Set B`, shown in Table 7.7. `System B.1` is the system that all the jobs have 400 time-units as their periods; then the period of the `LR Report Job` is, at first, reduced to 100 time-units in `System B.2`; the `UR Report Job` is the second in `System B.3`; eventually the `Conveyor Report Job` is the last in `System B.10`. Although, the order of decreasing these periods is arbitrary and the jobs will not appear equally on the four processors, the overhead on the processors is given objectively to them as much as possible. The motivation for `Study B` is to examine the influence by the appearances of a job rather than to only measure how much time is taken in each system. This is because the task execution times will be different on more powerful machines.

The `Study B` with the `System Set B` is also performed four times; a timing table has been prepared with the results of `Study B` (see Table 7.8). It is noted that the times on the table are slight different every attempt but the differences are negligible for the same reasons as `Study A`. According to the table, it is understood that the timing difference is slightly increased even if it is small, in particular, in the beginning of the

System Set B; however the increase is steady in the System B.5, B.6, and B.7 because only a single task belongs to the LR Move Job, UR Move Job and Conveyor Move Job, meaning that it makes less appearances of these jobs on the scheduling round; there is a strong increase in the end, in particular, between the System B.9 and B.10. Overall there is a 12.25 second difference between the System B.1 and B.10 and this difference is almost 2.5 times more than for System B.1. Through Study B, it is understood that whenever the number of the appearances of a job is increased in a scheduling round, it takes more time to generate schedules in the approach using the prototype toolset; in particular, this behaviour is strongly non-linear. Thus, it can be concluded that the number of the appearances of a job in a scheduling round influences the time to generate schedules; this is assumed to result from state explosion in verification.

Unit: Seconds

System	B.1	B.2	B.3	B.4	B.5	B.6	B.7	B.8	B.9	B.10
Attempt 1	8	8	9	10	11	11	12	13	14	20
Attempt 2	7	8	9	10	11	11	11	13	14	19
Attempt 3	7	8	9	9	11	11	11	12	14	20
Attempt 4	7	9	9	10	10	11	11	12	14	19
Average	7.25	8.25	9.00	9.75	10.75	11.00	11.25	12.50	14.00	19.50

Table 7.8: Time Required to Find Schedules for System Set B

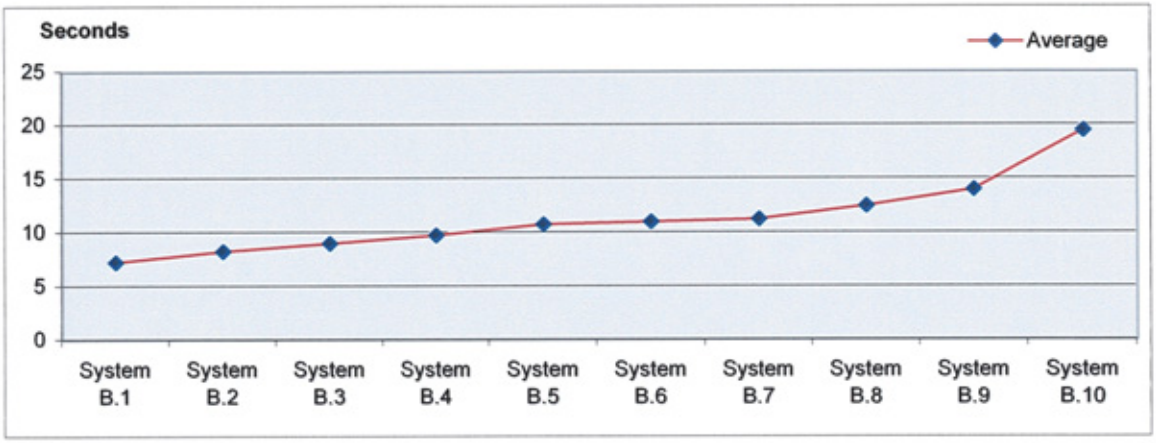


Figure 7.15: Average Schedule Time for System Set B

7.2.5.3 Influence of Utilisation of Processors and Networks – Study C

It is possible to compute processor utilisation and network utilisation from computation times and transfer times of tasks and messages, and their periods. For example, supposing that `Load Robot` in the Robot Transport system has only one task, `LR Actuator` and its period and computation time are assigned 50 and 10 time-units respectively, the utilisation of the `Load Robot` in percentage is 20% according to the utilisation formula (see the Chapter 2); in the case that the task is given 20 time-units, the utilisation will be 40%. So, the utilisation of the `Load Robot` can be increased if the `LR Actuator` is assigned more computation time.

Here, `Study C` investigates the influence between higher utilisation and lower utilisation in the proposed approach. With the jobs on the Robot Transport system, the computation times and transfer times for all the tasks and messages are assigned 5 time-units at first; then the assigned times are increased in steps by 2 time-units. In other words, the next system will have more utilisation than the previous as all the computation times and transfer times are allocated with more time.

CP: Control Panel **CON: Conveyor** **LR: Load Robot** **UR: Unload Robot**

	Task Computation Time	Processors Utilisation				Message Transfer Time	Network Utilisation
		CP	CON	LR	UR		
System C.1	5	20.0%	17.5%	18.7%	18.7%	5	22.5%
System C.2	7	28.0%	24.4%	26.2%	26.2%	7	31.5%
System C.3	9	36.0%	31.5%	33.7%	33.7%	9	40.5%
System C.4	11	44.0%	38.5%	41.2%	41.2%	11	49.5%
System C.5	13	51.9%	45.4%	48.7%	48.7%	13	58.5%
System C.6	15	59.9%	52.4%	56.2%	56.2%	15	67.5%

Table 7.9: System Set C

Consider the systems, `System Set C`, shown in Table 7.9 in which each system has different utilisations on the processors and networks because different computation times and transfer times in each system are allocated to the tasks and messages. The assigned times are increased up to 15 time-units and so `System C.1` has the least utilisation and `System C.6` has the greatest. As usual, the `Study C` has been performed

four times and measured the timing consumptions of each system; the result of the `Study C` is shown in Table 7.10. When the assigned times to the tasks and the message on a system are increased, it takes slightly more time to produce schedules in the approach. Comparing the `System C.1` and the `System C.6` which are assigned to 5 time-units and 15 time-units respectively on their tasks and messages, the difference is around 5 seconds. This is not a considerable difference but at least it can be concluded that computing schedules for systems with higher utilisation on processors and networks demands more time than systems with lower utilisation.

Unit: Seconds

	System C.1	System C.2	System C.3	System C.4	System C.5	System C.6
Attempt 1	13	14	14	15	16	18
Attempt 2	13	13	14	15	17	20
Attempt 3	13	14	14	14	16	18
Attempt 4	13	13	14	14	16	18
Average	13.00	13.50	14.00	14.50	16.25	18.50

Table 7.10: Time Required to Find Schedules for System Set C

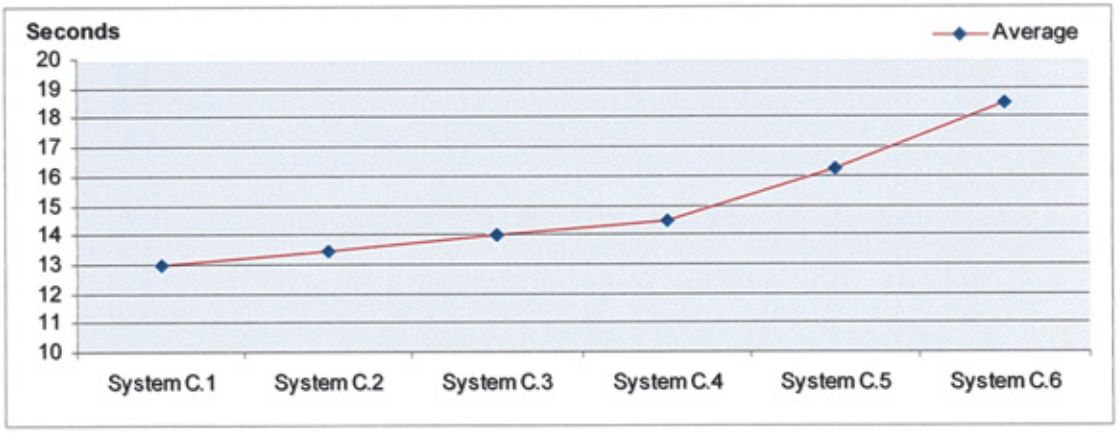


Figure 7.16: Average Schedule Time for System Set C

Although this study has been performed with the computation times and transfer time up to 15 time-units, it can be extended by increasing time-units further and so find the most utilisation in the `Study C`. This additional study confirms that:

- If a schedule exists, it must be identified.

- If a schedule does not exist, no schedule must be identified.

There are additional 4 systems in the `System Set C`. This time, each system will have its computation times increased by 1 time-unit from the last system, `System C.6`, in order to examine these systems cautiously, so the assigned times to the tasks and messages will be up to 19 time-units. Table 7.11 shows whether a feasible schedule is found for each system in the approach and includes the utilisations on the processors and network. According to the Table, a schedule is successfully found up to 18 time-units but there is a failure to deliver a schedule with 19 time-units. It means that the latest successful system, `System C.9`, whose computation times and transfer times on the tasks and messages are 18 time-units will have the most utilisation bound: 72.0% utilisation bound on the `Control Panel`, 63.0% on the `Conveyor`, 67.5% on the `Load Robot` and 67.5% on the `Unload Robot`, and also 81.0% on the `Ttp` Network. It is noticed that these utilisations on the processors and network are applicable to this study in which the same timing value is given to the tasks and messages; otherwise the utilisations will be different.

Although the approach does not show why `System C.10` cannot be computed, at least it shows whether or not a schedule exists. Practically speaking, it is extremely difficult to identify a schedule by hand in such complicated systems, even the `System Set C`. In particular, when various timing values are allocated to the tasks and messages, the benefits of this approach are more notable.

CP: Control Panel **CON: Conveyor** **LR: Load Robot** **UR: Unload Robot**

Schedule Existence		Task Computation Time	Processors Utilisation				Message Transfer Time	Network Utilisation
			CP	CON	LR	UR		
System C.7	O	16	64.0%	56.0%	60.0%	60.0%	16	72.0%
System C.8	O	17	68.0%	59.9%	63.7%	63.7%	17	76.5%
System C.9	O	18	72.0%	63.0%	67.5%	67.5%	18	81.0%
System C.10	X	19	75.9%	66.5%	71.2%	71.2%	19	85.5%

Table 7.11: Additional System Set C

7.2.5.4 Influence of Different Timing Values – Study D

Study D examines the influence of different timing values on tasks and messages based on the conclusion of the Study C. Consider the systems, System Set D, in which the different but less computation time are assigned to the tasks, comparing with the System Set C. For example, all the tasks on the Control Panel in the System C.1 have 5 time-units for their computation times, whereas the tasks on the Control Panel in the System D.1 will have less or equal to 5 time-units such as one of 1, 2, 3, 4, 5 time-units but each task will be different. So, the processors in the System Set D will be lower utilisation than the processors in the System Set C but have various computation times assigned to their tasks (see Table 7.12).

CP: Control Panel **CON: Conveyor** **LR: Load Robot** **UR: Unload Robot**

	Task Computation Time	**Processors Utilisation**				**Message Transfer Time**	**Network Utilisation**
		CP	**CON**	**LR**	**UR**		
System D.1	1, 2, 3, 4, 5	12.5%	9.0%	11.2%	11.2%	5	22.5%
System D.2	3, 4, 5, 6, 7	20.4%	16.0%	18.7%	18.7%	7	31.5%
System D.3	5, 6, 7, 8, 9	28.5%	22.9%	26.2%	26.2%	9	40.5%
System D.4	7, 8, 9, 10, 11	36.5%	29.9%	33.7%	33.7%	11	49.5%
System D.5	9, 10, 11, 12, 13	44.4%	37.0%	41.2%	41.2%	13	58.5%
System D.6	11, 12, 13, 14, 15	52.4%	44.0%	48.7%	48.7%	15	67.5%

Table 7.12: System Set D

The conclusion of the Study C is known that systems with higher utilisation on processors and networks require more time to find schedules than systems with lower utilisation. However, the conclusion is not generally true according to the Study D. When comparing the timing consumption between the System D.1 and the system C.1, the System D.1 takes more time than the System C.1 even though the processors in the System D.1 have lower utilisation than the processors in the System C.1; even the System D.1 takes more time than System C.6 required the greatest time among the systems in the System Set C; the time difference between the System D.6 and the System C.6 is dramatic as it is about 24 seconds. Accordingly, it is clear that the

conclusion from the Study C cannot always be guaranteed but there is another conclusion brought from the Study D. The systems with the different timing values to tasks and messages take more time than the systems with the same timing values, even if the former systems have less utilisation than the latter. Thus, it can be concluded that assigning the different timing values to tasks and messages has more influence than assigning higher utilisation in the proposed approach. However, the conclusion from the Study C is still operative in the System Set D as the System D.6 demands more time than other previous systems, but there is an exception because the System D.4 does not require less time than the System D.3. The exception may be the same reason as the conclusion of the Study D; it is also assumed the results from state explosion as the different timing values may increase more states than the same timing values in verification.

Unit: Seconds

	System D.1	System D.2	System D.3	System D.4	System D.5	System D.6
Attempt 1	19	21	32	24	39	41
Attempt 2	19	21	33	22	39	44
Attempt 3	20	22	33	23	38	44
Attempt 4	20	22	32	22	42	41
Average	19.50	21.50	32.50	22.75	39.50	42.50

Table 7.13: Time Required to Find Schedules for System Set D

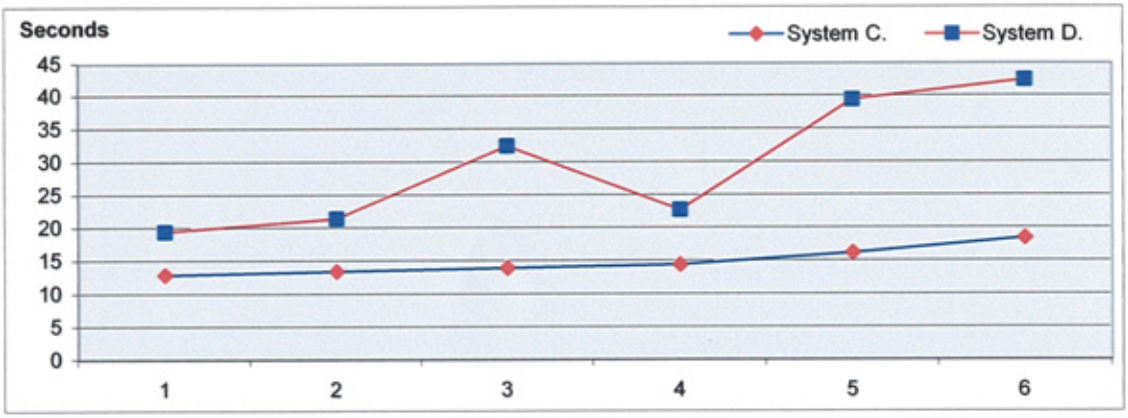


Figure 7.17: Average Schedule Time for System Set C and D

7.3 Summary

Using the prototype toolset introduced in the previous chapter, two case studies with the Adaptive Cruise Control system and Robot Transport have been presented to demonstrate the applicability and usability of the proposed approach. These case studies are of systems typically found in automobile and manufacturing industry. The descriptions and operations of these systems have been presented to adapt the systems into the time-triggered architecture, and the required operations for the systems on the architecture have been described in detail. In particular, The first case study with the ACC system have been focused on illustrating each step of analysing the system using the prototype toolset, and the generated scheduling result for the system has been introduced. In the second study, timing parameters has been varied with four studies to find factors which may affect the approach when generating schedules. The results of those studies allow the following conclusions:

- Arbitrary times assigned to tasks, messages and jobs in a system such as their computation times, transfer times, periods, etc, do not have any influence in the proposed approach to generate schedules. Consequently, any arbitrary time-units can be considered such as a second or millisecond, etc.
- Whenever the number of appearances of a job in a scheduling round is increased, it takes more time to generate schedules and this behaviour is strongly non linear. This is assumed to result from state explosion in verification.
- In general, systems with higher utilisation on processors and networks demand more time to generate schedules than systems with lower utilisation. However, this is not always guaranteed when systems have the tasks and messages allocated with different timing values individually. In this case that the systems with lower utilisation demand more time to generate schedule than systems with higher utilisation. It is understood that assigning the different timing values to each task and message in systems has more influence than assigning higher utilisation.

Chapter 8

Further Schedule Constraints

One of the considerable advantages of the proposed approach, using timed automata models, is the possibility of applying further constraints to timed automata models in order to yield schedules with additional properties. Besides, it is also possible to adapt the timed automata to the different types of scheduling generation easily, depending on the schedule constraints on systems. These are certainly different from the existing mathematical methods and algorithms. In this chapter, further schedule constraints such as jitter, schedule compaction and precedence constraints between jobs on schedules are considered and the ways to simply apply the constraints with the models and properties are introduced.

8.1 Schedules Considering Jitter

The issue of jitter on schedules was considered with the schedule of the FC system generated in the chapter 6. According to the schedule which depicts the `Control` job with a blue colour and the `Alarm` job with a yellow colour shown again in Figure 8.1, the `Control` job having 100 time-units period starts at 0 with `Sample` task and finishes at 50 with `Actuator` task; the `Alarm` job starts a little late at 10 with `Alarm` task in order to wait for the resource of `Plant` processor already occupied by the `Sample` task in the `Control` job, and ends up with `indicator` task at 40. Soon after, the `Alarm` job starts again at 50 time-units because its period is 50 time-units. But, this time the job can start the `Sample` task at 50 time-units because there is no disturbance from the `Control` job whose work has already finished, and so it can finish its work at 70 time-units. The

schedule for the two jobs can make to complete successfully the tasks and messages within its period; thus it will be straightforward to adapt the schedule for the FC system.

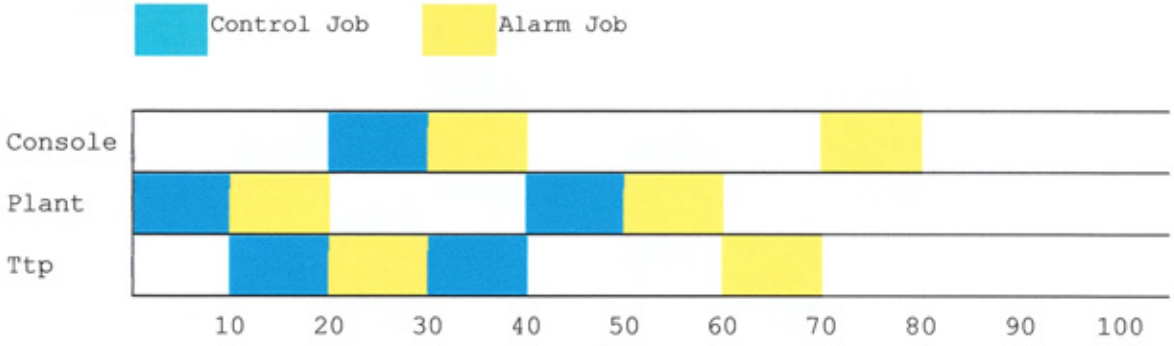


Figure 8.1: The Generated Schedule for FC System

However, there is an unpredictable time between the first period and the second period of the `Alarm` job in the schedule for the FC system, caused by the different starting time such that the job starts at 10 time-units in the first period and at 50 time-units in the second period. This means that the `Alarm` job misses a regular interval and consequently it may lead to instability. This unpredictable time is called jitter [But97, LH96, MFF+01, NS00] which is the deviation of two consecutive arrival instances. Considering a task as a set of period instances, the jitter of the *k*th task within a given job, $Jitter(t_{i,k})$, is defined as follows:

$$Jitter(t_{i,k}) = \left| start(t_{i,k}) - start(t_{i,k-1}) - t_i.p \right| \text{ for } k = 1,2,3,...$$

where $t_i.p$ is a period of the task t_i and *start*() is a function indicating the start time. However, this research is currently considering the jitter of a job and so the jitter of the *k*th job, $Jitter(J_{i,k})$, is defined:

$$Jitter(J_{i,k}) = \left| start(J_{i,k}) - start(J_{i,k-1}) - J_i.p \right| \text{ for } k = 1,2,3,...$$

Many real-time systems such as avionic systems, automated factories, automotive systems, etc, require jitter to be strictly limited, though this may be difficult to achieve as a result of occasionally accessing shared resource, precedence constraints between tasks, etc. These concerns over the jitter problem in those systems arise for many reasons: fault tolerance, avoiding unsafe operations, discovering faults and so on. In

particular, the extra times in earlier fault detection and recovery could influence the difference between a success and disaster in safety-critical systems [LH96]. Considering the generated schedule for the FC system, which exhibits a jitter problem, with safety-critical implication, the proposed approach is further adapted to produce a schedule, in particular, with no jitter.

Two types of schedule are expected for the FC system considering no jitter, shown in Figure 8.2. The first one is that the `Alarm` job in the second period intentionally has 10 time-units delay, in order to make the same starting time as the first period. So the job starts at 60 time-units rather than at 50 time-units. In this case there is no jitter in the `Alarm` job, and the job will have the exact same pattern every period.

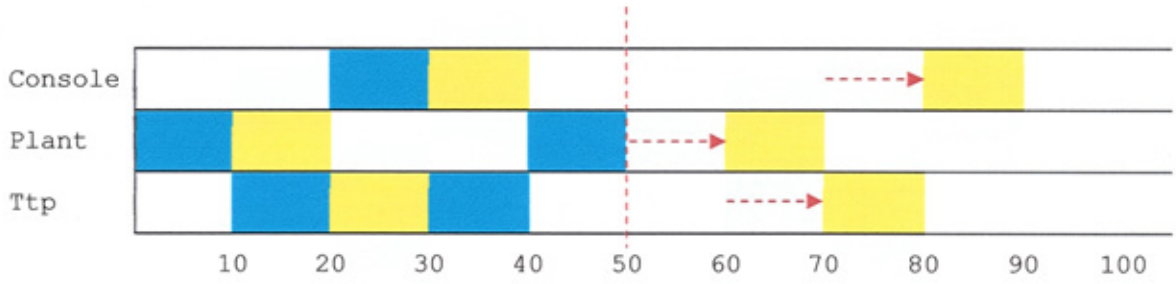


Figure 8.2-1: The Schedule for FC System Considering No Jitter

The second is that the `Alarm` job starts earlier than the `Control` job. This means that the `Alarm` job in the first period starts at 0 and still remains the same time, 50 time-units, in the second period. Consequently, there is no jitter in the `Alarm` job. However, in order to create no jitter for the `Alarm` job, the early start time of the `Control` job will have to be sacrificed in its schedule by its starting time being delayed at 10 time-units. In particular, the last task of the `Control` job has additional 10 time-units delay, disturbed by the `Alarm` job. Nevertheless, the `Control` job is still able to finish within its period which is 100 time-units.

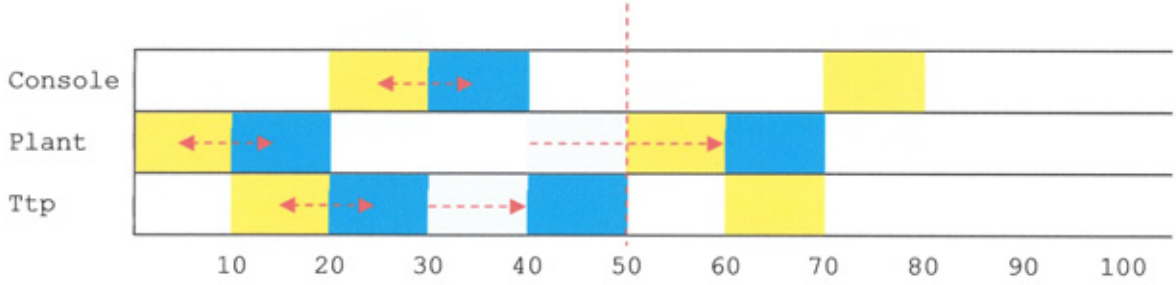


Figure 8.2-2: The Schedule for FC System Considering No Jitter

Consequently, it is obvious that all the instances within a schedule occur simply at their exact period, in order to generate a schedule with no jitter, $S_{\overline{jitter}}$. A schedule with no jitter within a scheduling round can be defined:

$$\forall_i \ \ start\left(J_{i,k}\right) - start\left(J_{i,k-1}\right) == J_i.p \text{ and } \ start\left(J_{i,k}\right) < \mathbb{R} \text{ for } k = 1,2,3,...$$

where $\mathbb{R}$ is a scheduling round. Two methods are examined to synthesise the schedule based on the proposed approach: one is using a modified temporal property, Method A, and the other uses a modified timed automata model, Method B.

8.1.1 Jitter Control with Temporal Properties – Method A

Jitter is constrained using a temporal property of the following form

$$\mathrm{E}\langle\rangle\left(\forall_i \ \ start\left(J_{i,k}\right) - start\left(J_{i,k-1}\right) == J_i.p\right) \text{ for } k = 1,2,3,...$$

It is possible to synthesise $S_{\overline{jitter}}$ using the UPPAAL verifier if the property is satisfied. To do this, firstly the time value (or clock value) has to be extracted from a clock variable, in order to get all the instances' starting times. Unfortunately, this is impossible in the UPPAAL tool which does not allow a clock value to be saved in an integer variable, but only allows the comparison of a clock variable with an integer variable. To overcome this problem of counting a clock variable, a new timed automata model, Timer, is introduced in Figure 8.3.

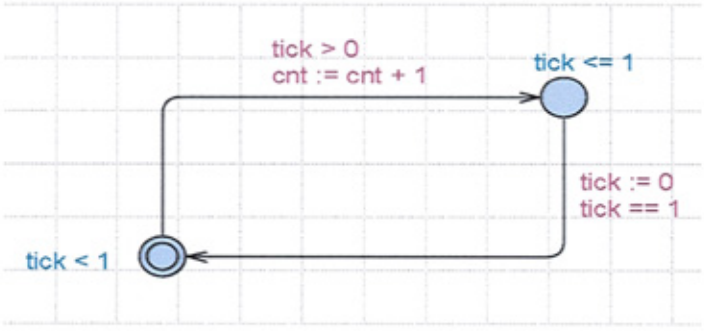


Figure 8.3: Timed Automata Model for Counting Clock Values

This model is simple. It has two states, two transitions, one clock variable, named `tick` and one global integer variable, named `cnt`. It is known that all clocks in timed

automata have the same clock rate and so every time the `tick` clock becomes 1, the `cnt` variable is added by 1 and then the `tick` clock is assigned to 0. Consequently, it is possible by counting a clock and referring to the `cnt` variable instead of a clock. However, the `cnt` should be added before the `tick` clock is equal to 1, in order to avoid miscounting of the clock. For example, suppose that all clocks in models are at 9 time-units and then at 10 time-units there are only two transitions available, including the one that increases the `cnt`. If a verifier chooses the other transition rather than the transition for the increase, the `cnt` is not properly evaluated as it is still 9. In this case, choosing the transition for increasing the `cnt` has to be first and then selecting the transition is second. This is reason why it is necessary to increase the `cnt` slightly earlier than other transitions.

With the clock model, it is possible to save the starting time of a job. The model for a job includes saving the `cnt` value into an integer array, `sTime`, indexed by `i` initially with 0 whenever the model starts with a transition leaving the `idle` state. Of course, the index has to be increased in order to save the next starting time of the model. Figure 8.4 shows the change to the `Alarm` job model in the FC system in order to measure time.

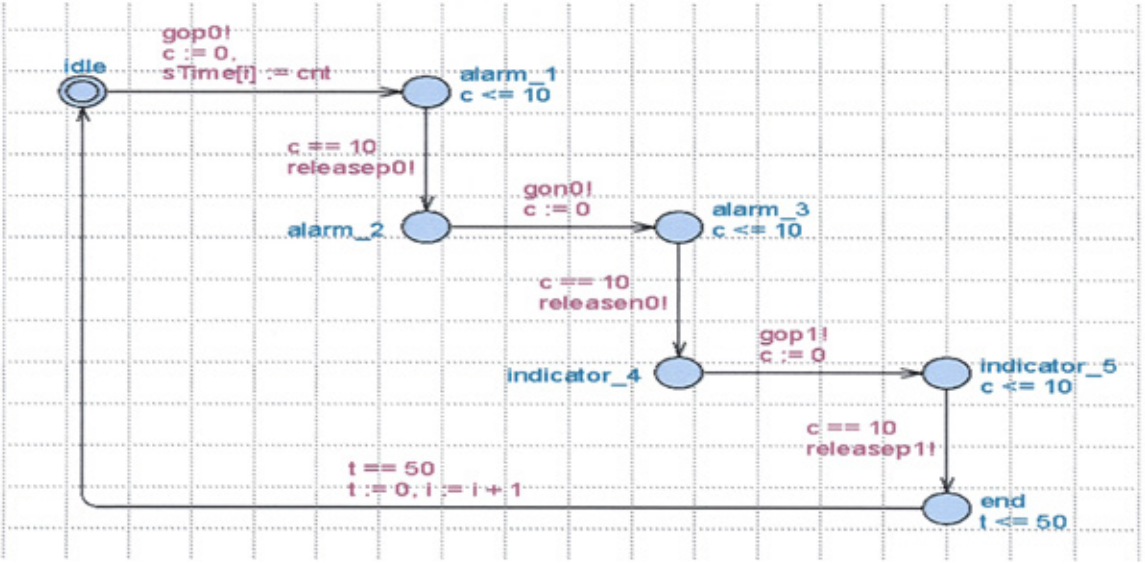


Figure 8.4: The Job Model for No Jitter

```
E<>(
((Alarm_Job.sTime[Alarm_Job.i] - Alarm_Job.sTime[Alarm_Job.i-1] == 50)
  and Alarm_Job.i >= 1 and Alarm_Job.end)
  and Control_Job.end and Time <= 100
)
```

Figure 8.5: A Temporal Logic Property for No Jitter

Consider the FC system again to retrieve a schedule with no jitter using this approach. The time difference between two consecutive instances of the `Alarm` job, $start(\texttt{Alarm\_Job}_k) - start(\texttt{Alarm\_Job}_{k-1})$, should be 50 time-units as its period. In a temporal logic property, the values of the `sTime` array are referred as follows:

```
Alarm_Job.sTime[Alarm_Job.i] - Alarm_Job.sTime[Alarm_Job.i-1] == 50
```

where `Alarm_Job.i` is equal to greater than 1. There is no need to consider any jitter problem in the `Control` job because it appears only once within the scheduling round, 100 time-units. The complete property is shown in Figure 8.5, imposing a schedule with no jitter for the FC system.

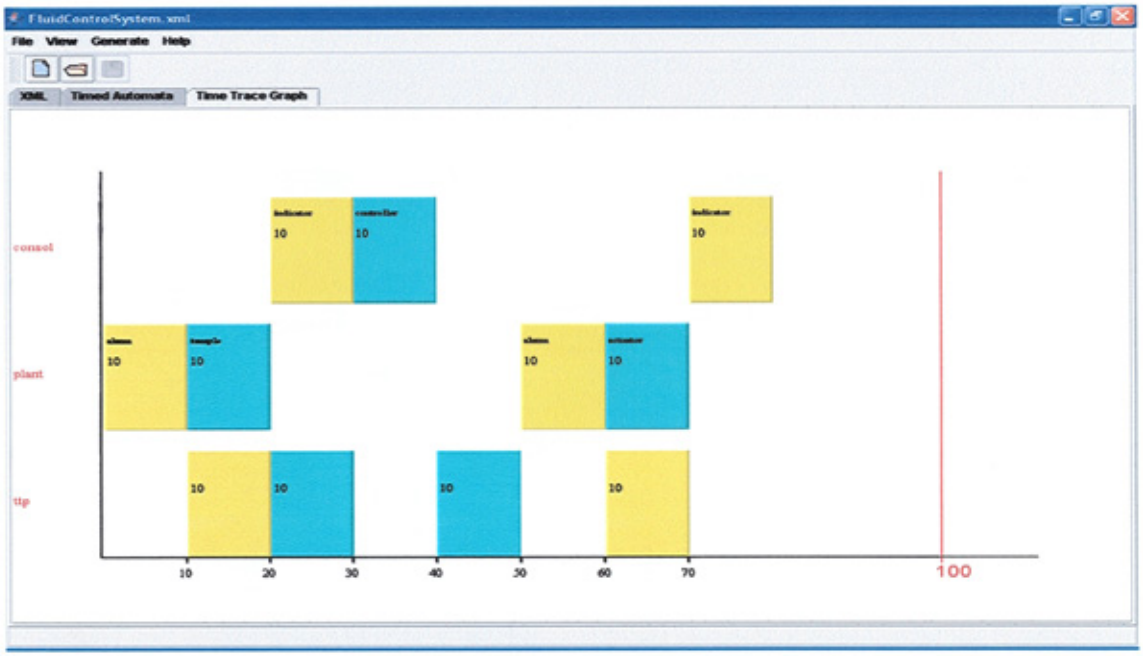


Figure 8.6: The Schedule with No Jitter in Alarm Job using Method A

Unfortunately, the current prototype toolset does not address the jitter problem when generating UPPAAL models and properties. Future work will include this work

with a high priority. However, it is possible to manually apply the models and include the property in the toolset; Figure 8.6 displays a schedule for the FC system manually applied. According to the schedule, the `Alarm` job starts at 0 and 50 time-units, meaning that the job has occurred at the exact point as its period. So there is no jitter problem in the job. But the `Control` job is slightly delayed as expected. This schedule looks like the schedule shown in the Figure 8.2-2 which is expected for the FC system with no jitter. Consequently, it shows the possibility of the approach in generating a schedule with no jitter using a temporal logic property.

8.1.2 Jitter Control with UPPAAL Models – Method B

By creating a job model which already exhibits the jitter problem, it is also possible to retrieve $S_{\overline{jitter}}$. It is known that a schedule with no jitter for a job should fulfil $start(J_{i,k}) - start(J_{i,k-1}) == J_i.p$. In other words, the starting time of the kth instance is equal to the sum of the starting time of the first instance and $(k-1)$ multiplies of the period of a job, defined by:

$$start(J_{i,k}) = start(J_{i,1}) + (J_i.p \times (k-1)) \text{ for } k = 1,2,3,...$$

For example, if there is a job with a period of 50 time-units and 10 time-units as the starting time of its first instance, the $start(J_{1,5})$ is expected to be 210 time-units. Based on this fact, then $S_{\overline{jitter}}$ within a scheduling round is also defined as follows:

$$\forall_i \; start(J_{i,k}) == start(J_{i,1}) + (J_i.p \times (k-1)) \text{ and } start(J_{i,k}) < \mathbb{R} \text{ for } k = 1,2,3,...$$

With the clock model already introduced, the job model is amended to make the starting times of all the instances to be multiples of the first instance. To do this, an urgent location and a guard are added to the model, ensuring that the model waits as much as its period before the second (or remaining) instance starts. So, it can enforce the starting times of all instances to be fixed. However, there is time delay problem because the model has to start immediately with no delay after waiting for its period. This problem is solved with using an urgent location [LPY97, PL00] provided in UPPAAL. In this location marked with **U**, the passage of time is not allowed but communication is possible with other locations by a channel, this is only the difference between a

committed and urgent location in UPPAAL as a committed location does not allow even any communication. Figure 8.7 shows the amended timed automata model of the `Alarm` job in the FC system.

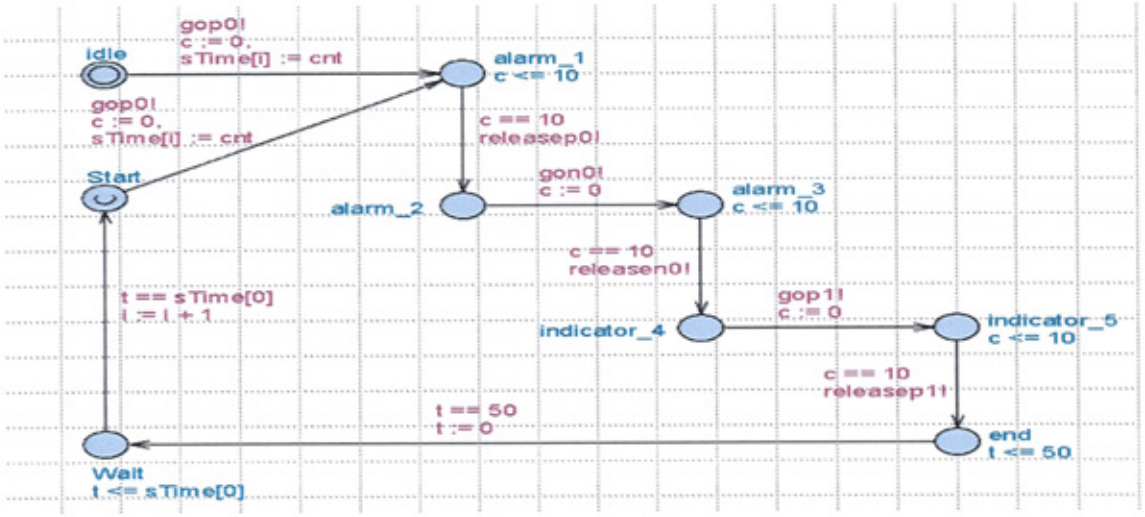


Figure 8.7: The Job Model for No Jitter

To briefly explain Figure 8.7, the model has two additional locations: `Wait` and `Start`. The former is used to wait for `sTime[0]` time, which is the starting time of the first instance, and the latter as an urgent location is to start the next instance immediately such that when reaching the urgent location, the transition between the `Start` and `alarm_1` will be fired immediately with no delay. This model saves the times of the remaining instances into the integer array, `sTime`.

When also manually applying the models for the FC system into the toolset, the generated schedule for the system is included in Figure 8.8. According to the schedule, the `Alarm` job starts at 10 and 60 time-units, and the `Control` job starts at time 0. It means that this schedule also includes no jitter in the `Alarm` job. The schedule also looks like the schedule in the Figure 8.2-1 which generates a schedule with no jitter for the FC system. Consequently, it is also possible to generate a schedule with no jitter by amending the timed automata models in the approach.

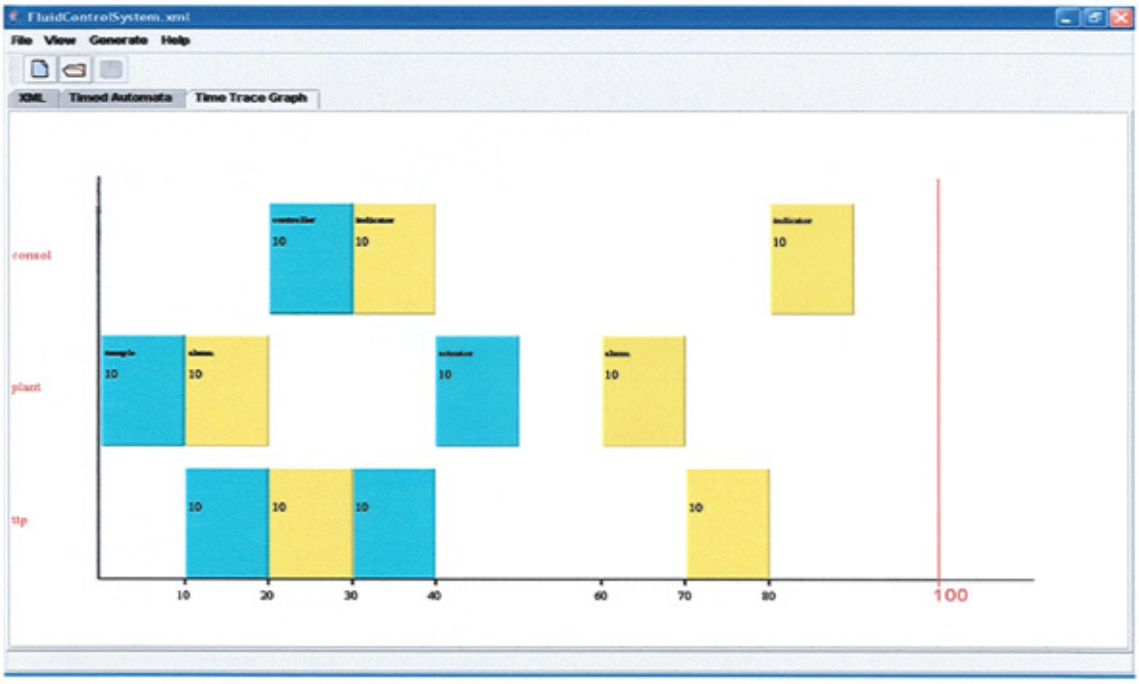


Figure 8.8: The Schedule with No Jitter in Alarm Job using Method B

8.1.3 Comparison between using Temporal Properties and UPPAAL Models

There is certainly a difference between using a temporal property (Method A) and UPPAAL models (Method B) in generating a schedule with no jitter. Here, the time taken to produce a schedule with no jitter using the Methods has been measured and compared, based on the computer environment which works with CPU 2.20GHz and 512MB of RAM on Windows OS. Starting with the scheduling round, 100 time-units, which is the least common multiple of the `Alarm` job and the `Control` job period, the measurement has been performed and then the round has been increased by 100 time-units for the next measurement until reaching 800 time-units. Figure 8.9 represents the result between them.

According to the Figure, there is little difference between the two methods within the rounds from 100 to 300 time-units. However, it is easily recognised that the Method B takes less time than Method A after 400 time-units. The difference is getting clear when the round time is being increased because as the behaviour of Method A is linear whereas the behaviour of the Method B appears a geometric progression. This result is assumed that the Method A suffers more severely from a state explosion problem as it is based on exhaustive state space search. Thus, when considering producing a schedule

with no jitter from the approach, the Method B is a more efficient choice than the Method A. However, the model introduced in Method B is only the case of producing a schedule with no jitter but the model in Method B can be applied not only in a jitter problem case but also in more general cases by using various properties.

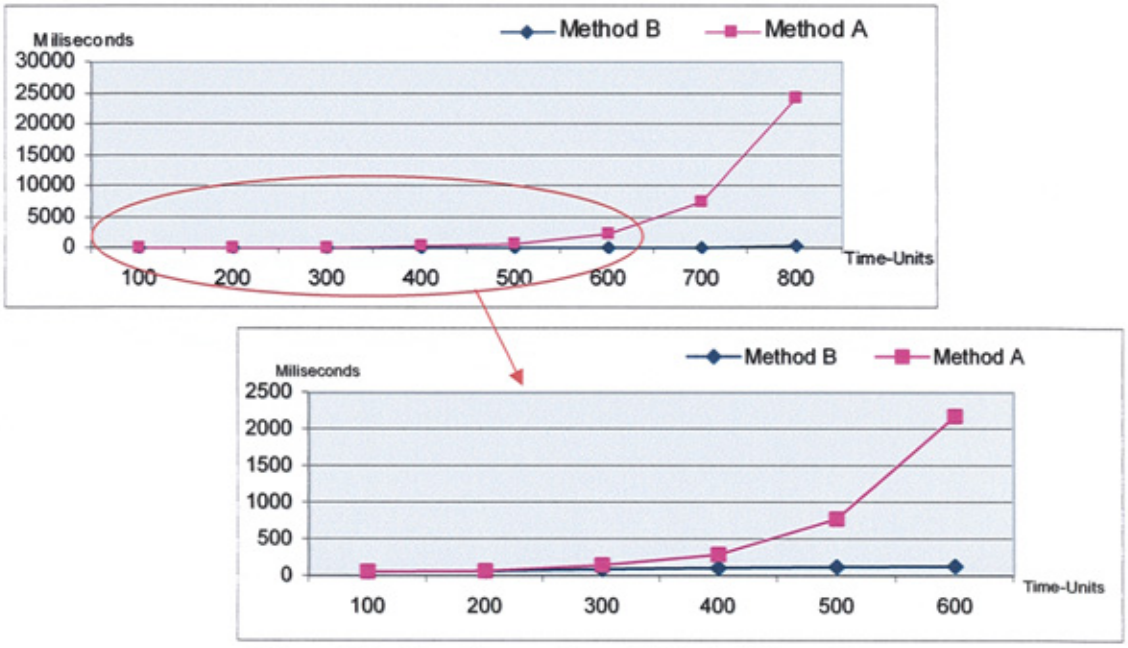


Figure 8.9: The Comparison of Method A and B

8.2 Scheduling Compaction

As a time-triggered architecture is based on the pre-determined points in time, a system using the architecture requires a static schedule. The behaviour of each task and message in the system can be predicted precisely within the schedule. On the other hand, there is a difficulty to extend a new task or a job to the system. If an extension is within the existing schedule, i.e. new tasks do not affect other tasks in the schedule, it will be acceptable. Otherwise, the static schedule has to be recalculated and reallocated. In particular, in a case that the schedule is fragmented, the extension is more difficult and certainly requires more effort to extend the system every time [Kop93].

This extensibility problem in a time-triggered architecture can be overcome by providing sufficient timing space in a scheduling round for future extensions in advance and thus the extensions will be easily accommodated in the space without disturbing the existing schedule. A scheduling compaction is a possible technique to provide sufficient

timing space and it is, in particular, a useful technique when a schedule is fragmented. The technique is based on compressing the fragmented schedule as much as possible and then secures a full timing space in a scheduling round.

```
Let M, TCP, TF be UPPAAL models, a property and a trace file respectively
Let isSatisfied ∈ {true, false} have false
Let Time be a sufficient time value to satisfy the UPPAAL models

do until isSatisfied
    TCP = CreateProperty( M, Time );
    if M |= TCP then
        Last_TCP = TCP ;
        Time = Time - AMOUNT ;
    else
        TF = GenerateTrace( M, Last_TCP )
        isSatisfied = true
    end if
end do
return TF
```

Figure 8.10: Pseudo Code for Scheduling Compaction

The scheduling compaction is easily applicable based on the proposed approach. From the approach, it is possible to retrieve many feasible schedules S_{ch} but not guarantee compacted schedules – which may be mostly already compact. It is only sure that they satisfy the constraints of a system. Thus, the additional utilisable way of the approach is introduced to retrieving compacted schedules. Based on the fact that feasible schedules are produced when the timed automata models generated from the approach satisfy with the generated property, the additional approach applies the fact iteratively. Ideally, it is assumed that a successful schedule is already produced with the property including a scheduling time. Then, the additional approach reduces a slight amount time from the scheduling time in the property and tries to verify the models with the changed property again. This work is iterated until the verifier fails to produce a schedule. Consequently, it is expected that the latest successful property enable to produce the compacted schedules, comparing with other successful properties. Figure 8.10 shows a pseudo code for this approach.

Unit: Time-Units

	Computation Time	Transfer Time	Period
Job A	10	10	150
Job B	15	10	150
Job C	20	10	150
Job D	25	10	150

Table 8.1: The Detail of the Jobs in System F

Consider the following system, `System F`, shown in Figure 8.11, which consists of two processors (`Processor 1` and `Processor 2`) and one network as an experimental study of retrieving a compacted schedule. There are 4 jobs available in the `System F`, simply named from `Job A` to `Job D`. The assigned timing values for the system are listed in Table 8.1: the transfer times of all the messages are 10 time-units; the periods of all the jobs are 150 time-units, the same as their deadlines; the tasks belonging to the `Job A` are 10 time-units for their computation times; following the job, the tasks belonging to the `Job B` , `Job C` and `Job D` are gradually increased by 5 time-units and so they are 15, 20 and 25 time-units respectively. Also, the jobs in the system have slightly different behaviours such that the `Job A` and `Job D` have a similar behaviour but a different direction, and the `Job B` and `Job C` are a similar instance (see the Figure 8.11).

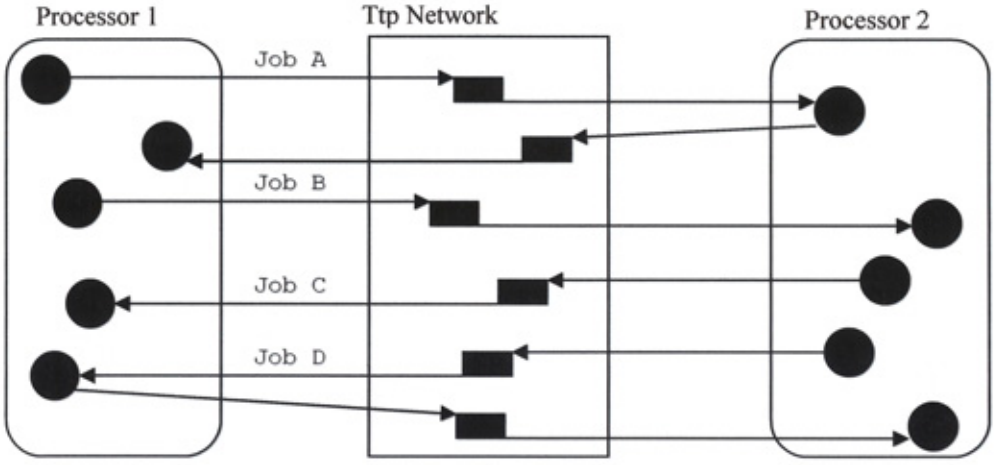


Figure 8.11: System F for Scheduling Compaction

It is assumed that a scheduling round is 150 time-units in the System F and that all the jobs have 150 time-units for their periods; within the scheduling round, many feasible schedules are possible when all the jobs start at the same time, 0 time-unit. It may be a laborious work to retrieve schedules manually but one of schedules, Schedule F-A, is depicted in Figure 8.12. This schedule can fulfil the timing constraints of the system such that all the jobs are finished within their deadlines (or periods). The Job A, B, C and D are finished at 135, 80, 70 and 145 time-units which are all within 150 time-units. Thus, the Schedule F-A can be acceptable for one of the successful schedules for the System F.

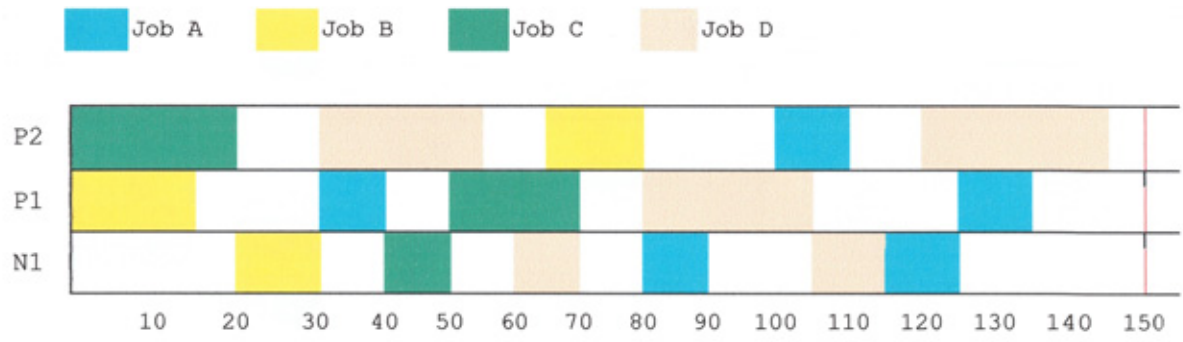


Figure 8.12: Schedule F-A, as One of the Schedules for the System F

Over the Schedule F-A, it is further assumed that two additional jobs, Job E and Job F, are expected for the experimental study of scheduling compaction. They also have the same timing values for their transfer times and periods as the existing jobs in the System F but are allocated with 20 time-units as their computation times for the tasks belonging to. The jobs have similar behaviour but a different direction shown in Figure 8.13.

When adding the two jobs into the Schedule F-A, at first it needs to find spaces for the tasks belonging to the jobs, and so both the Processor 1 and Processor 2 require 40 time-units for the tasks and the network needs 20 time-units. There are certainly spaces for the tasks that the Processor 2 has 55 time-units available; the Processor 1 has 70 time-units; the network has 90 time-units. However, these spaces are not contiguous but are fragmented such as the Processor 2 is available between 20 and 30, between 55 and 65, between 80 and 100, between 110 and 120, and between 145 and

150. There available times are insufficient for the additional jobs which require at least 20 time-units continuously for their tasks. Still, these may be possible to add only one of the jobs between 80 and 100 but this is also not enough over the `Schedule F-A`. Although there are spaces for the tasks of the `Job E` between 80 and 100 in the `Processor 2` and between 105 and 125 in the `Processor 1`, there is only 5 time-units available between two spaces for message passing, actually required 10 time-units for the message. Consequently, the `Schedule F-A` are not sufficient to add the additional two jobs and the schedule of the `System F` has to be entirely recalculated with the jobs.

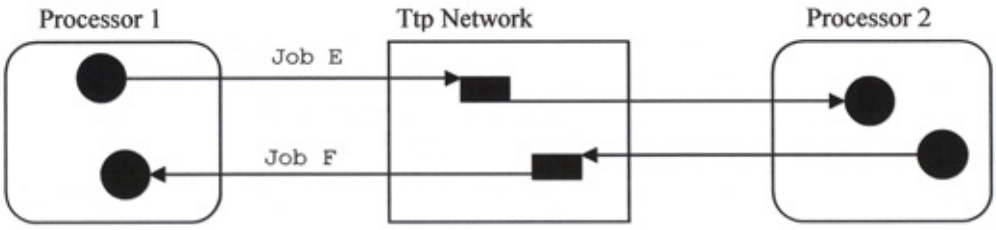


Figure 8.13: The Additional Two Jobs in System F

	Computation Time	Transfer Time	Period
Job E	20	10	150
Job F	20	10	150

Table 8.2: The Detail of Additional Two Jobs in System F

However, with the additional approach proposed above, it is possible to retrieve a compact schedule for the `System F`. It is already known that a schedule can be produced within the scheduling round, 150 time-units. Then, the additional approach reduces the round by 5 time-units in the generated property and tries to re-produce a schedule again. This work is iterated until it meets a failure. The reduction value, 5 time-units, is based on the computation times of all the tasks which are one of 10, 15, 20 and 25 time-units and also known as the greatest common divisor. However, there will be overhead when working with a small reduction value which increases the number of re-producing schedules until a failure. In this case, it is recommend that the value should be decided manually or some certain scheduling rounds have to be omitted to check whether a schedule is produced or not such as with 145, 140, 135, 130, 125 time-units, etc.

In order to produce a compact schedule for the `System F`, this experiment starts assuming the property which has the scheduling round of 120 time-units. As soon as this successful schedule is identified, the round in the property is reduced to 115 time-units for the next. With `System F`, the successes continue until the round reaches to 95 time-units and eventually this experiment meets a failure at 90 time-units. It means that the property with 95 time-units for the scheduling round will be the latest property which can produce the most compact schedule. Figure 8.14 shows the schedule produced from the approach and Figure 8.15 is the schedule used by the toolset.

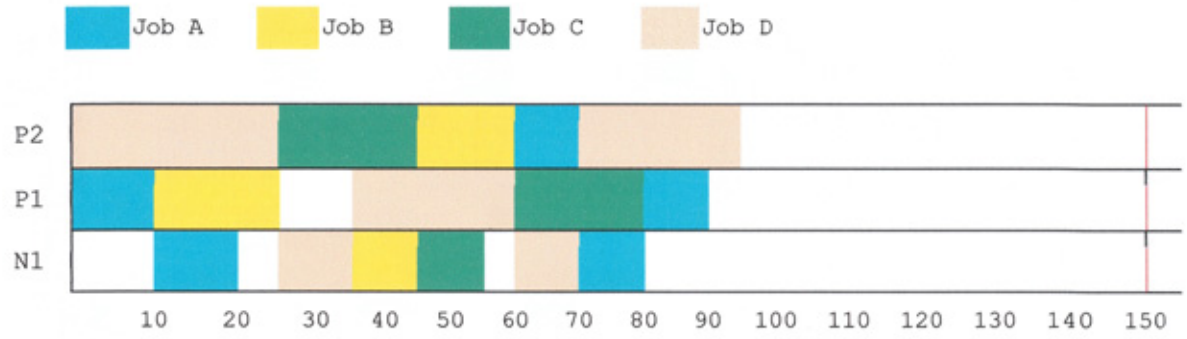


Figure 8.14: Schedule F-B, Produced for System F with Compaction

Figure 8.15: The Compact Schedule Generated from the Toolset

The schedule, Schedule F-B, is also one of successful schedules for the System F as all the jobs can be finished before their deadlines (or periods), and also has the same space available as the Schedule F-A. This time, however, the schedule is not fragmented but is mostly contiguous such as the Processor 2 has available spaces only between 95 and 150 time-units, and the Processor 1 is between 25 and 35, and between 90 and 150 time-units.

This newly available time will be sufficient for the additional jobs, Job E and Job F. Over the Schedule F-B, these additional jobs are added in the following way. The Job E starts at 95 time-units on the Processor 2; passes a message at 120 time-units; ends a task from 130 to 150 time-units on the Processor 1. The Job F firstly starts at 90 time-units on the Processor 1; passes a message at 110 time-units; eventually starts the last task at 120 time-units on the Processor 2. Figure 8.16 depicts the schedule with the two jobs.

Although the Schedule F-A and Schedule F-B have the same resource spaces available on the processors and network, the Schedule F-B is only acceptable to add the additional jobs as providing the contiguous resource spaces, without disturbing the existing schedules. Accordingly, this can reduce the overhead from recalculating the schedule. In this case, there is also a benefit in testability such that the overhead for testing the existing schedule does not required as it is already fitted in a system. And the test is needed for only the extension. Overall, it is concluded that this approach, which can produce a compact schedule, can help the overhead of future extension in a time-triggered architecture.

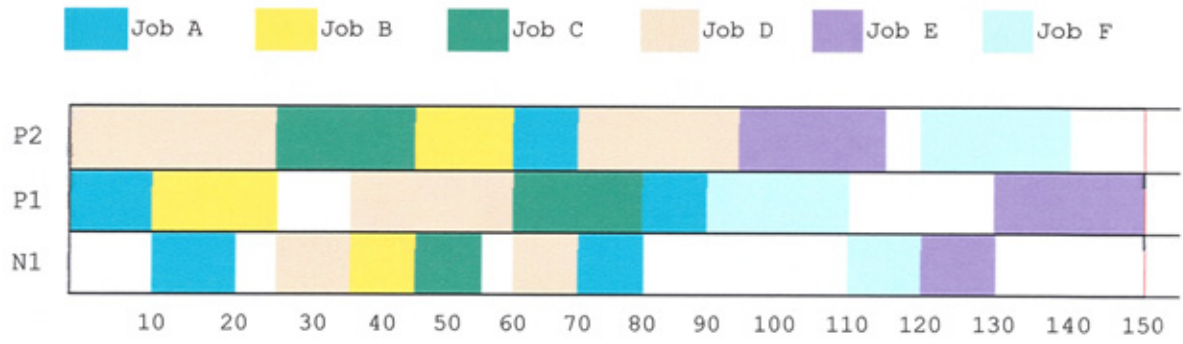


Figure 8.16: The Compact Schedule with Two Additional Jobs

8.3 Precedence Constraints between Jobs

In addition to the precedence constraints on tasks and messages, there is a further requirement concerning precedence constraints between jobs when generating a schedule for a system in the proposed approach. Reconsider the Robot Transport system again here. It is remembered that there are many jobs belonging to the system. In particular, among the jobs there are `CP-LR Command Job` and `LR Report Job` between the `Load Robot` and the `Control Panel`. The `CP-LR Command Job` is to give a command periodically from the `Control Panel` to the `Load Robot` in order to ask picking an item up on a reserved loading position; the `LR Report Job` is to periodically report the current status of the `Load Robot` and to display the status on the `Control Panel`. In the system, these two jobs are defined separately but it may be expected a precedence constraint between them. It is because the `Control Panel` may refer to the information from the `Load Robot` before giving a command to the `Load Robot`. In other words, the `CP-LR Command Job` may be affected by the `LR Report Job`. However, the timed automata model introduced in the approach previously does not have a functionality to give a precedence constraint between jobs. Thus, the generated schedule does not fit with this case properly in the Robot Transport system.

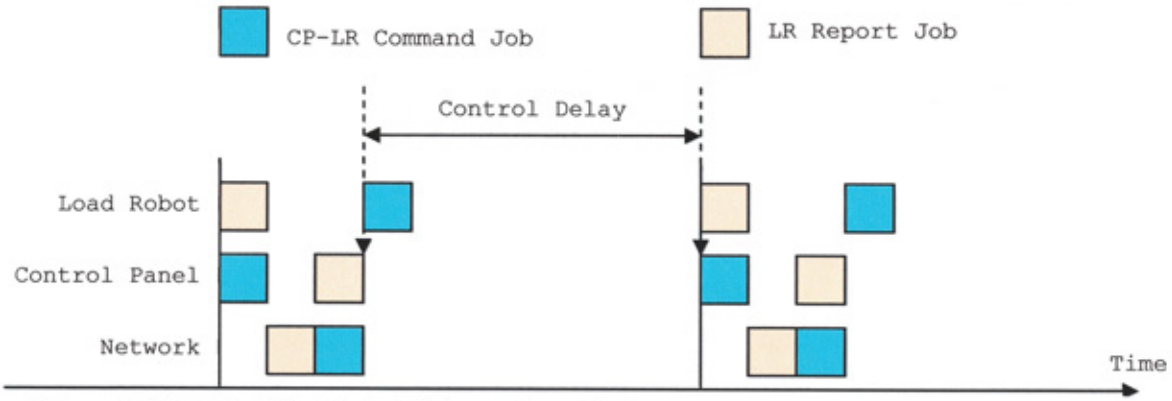


Figure 8.17: Schedule G-A, Without Considering Precedence Constraints between Jobs

Supposing that the two jobs are appeared once in a scheduling round, the schedule, `Schedule G-A`, shown in Figure 8.17 may be considered. In this schedule, as the `CP-LR Command Job` and `LR Report Job` starts simultaneously, the information of the `LR Report Job` cannot be reflected on the `CP-LR Command Job` immediately but has to be

waited for the next round in order to expect its reflection. In this case, the Load Robot may miss picking an item up but eventually pick it up some time later. It is expected a control delay between the two jobs; this delay should be as small as possible if a system requires fast reaction.

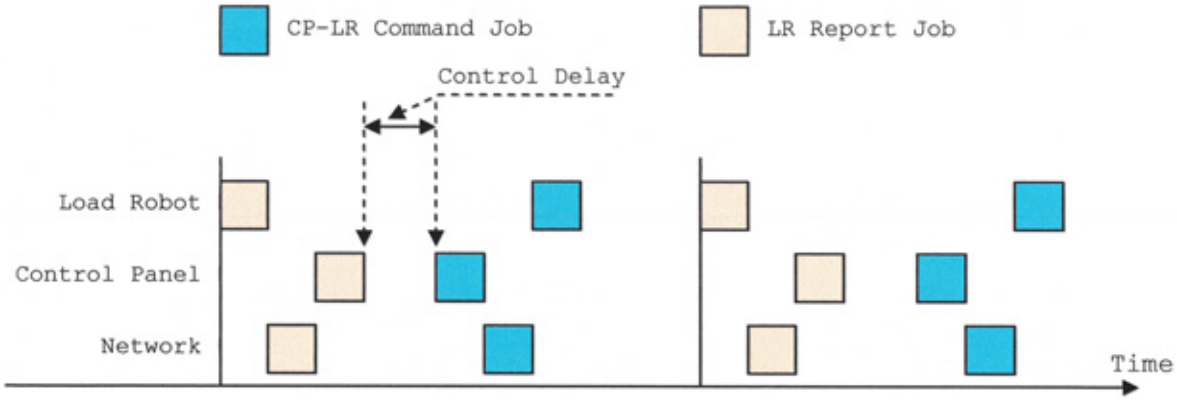


Figure 8.18: Schedule G-B, With Considering Precedence Constraints between Jobs

However, consider the schedule, Schedule G-B, in Figure 8.18. In this schedule, the LR Report Job starts and finishes first, followed by the CP-LR Command Job. And so the CP-LR Command Job can refer to the information from the LR Report Job immediately. The control delay will be smaller than the time to the next round and consequently the Schedule G-A will provide fast reaction than the Schedule G-B.

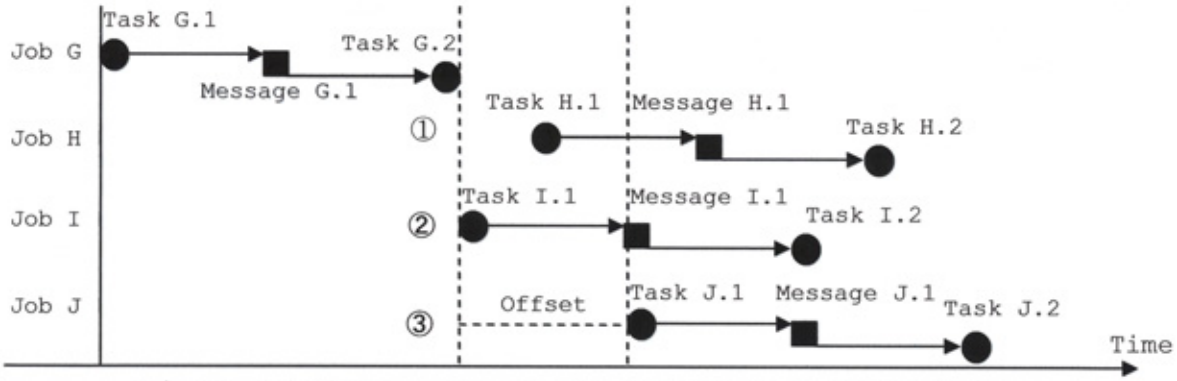


Figure 8.19: Three Types of Precedence Constraints between Jobs

Based on the timed automata models generated from approach, the improved automata models, which include precedence constraints between jobs, are introduced. Here, three cases of timed automata models are considerable depending on the types of precedence constraints between jobs. With the jobs shown in Figure 8.19, each type of

precedence constraint and each model of the timed automata are described in detail below. It is assumed that the jobs have the same period and the computation times of all the tasks and transfer times of all the messages are 10 time-units in the following automata models.

8.3.1 Precedence Constraint with Any Time – Case ①

First, a precedence constraint which enables to give an execution order between jobs is expected such that if there are two jobs, `Job G` and `Job H`, available in a system and the `Job G` starts first, the `Job H` has to wait until the `Job G` finishes its work and then the `Job H` will start later. It is important that there is not a specific time at which the next job, `Job H`, has to start but it is only sure that the time will be after the first job, `Job G`. This precedence constraint can be defined by applying such a flag technique between jobs, i.e. a job can make its progress only when a flag is raised up. Over the timed automata models for the `Job G` and `Job H` generated from the proposed approach, an integer (or boolean) variable `F` is declared for the technique with a default value 0 (or `false` in boolean); this variable is added to the end of the timed automata model for the `Job G` (after the `Tasks G.2`) as being assigned with other value, and the beginning of the timed automata model for the `Job H` (before the `Task H.1`) as being used a invariant on a transition. So, this is used as a flag between the models such that the `Job H` cannot start its work before the `Job G` raises the variable up such as assigning 1 (or `true` in boolean) into the variable. When the variable is set with such the value, the `Job H` is able to start its work some time later but there is no a specific time in the models. The timed automata model which includes this precedence constraint is shown in Figure 8.20.

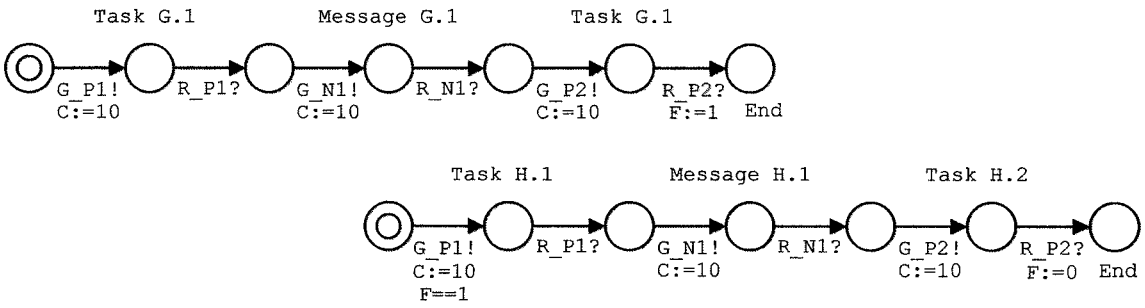


Figure 8.20: Timed Automata Model for Case ①

8.3.2 Precedence Constraint without Time Delay – Case ②

Second, a precedence constraint which enables to give an execution order without time delay between jobs is explored. Suppose that there are two jobs, `Job G` and `Job I`, in a system. This precedence constraint can force that the `Job G` starts first and then the `Job I` will start immediately after the completion of the `Job G`. On this occasion, there has to be no time delay between the first job, `Job G` and the next job, `Job I`. This precedence constraint can be defined by using the committed location, marked by **C** in UPPAAL. In the committed location, the passage of time and any interference from others are not allowed, and so the location has to be left immediately. Over the timed automata models for the `Job G` and `Job I` generated from the approach, extra committed locations are added to the end of the timed automata model for the `Job G` (after the `Tasks G.2`) and the beginning of the time automata model for the `Job I` (before the `Task I.1`). Then, there is an additional channel between these committed locations, named as `F_C`. This channel is used as such a flag between the models. And so the `Job I` cannot start its work before it receives a synchronisation from the `Job G` via the channel. It is known that as the synchronisation is over the committed locations, there will be no time delay between the jobs. The timed automata model applied this precedence constraint is shown in Figure 8.21.

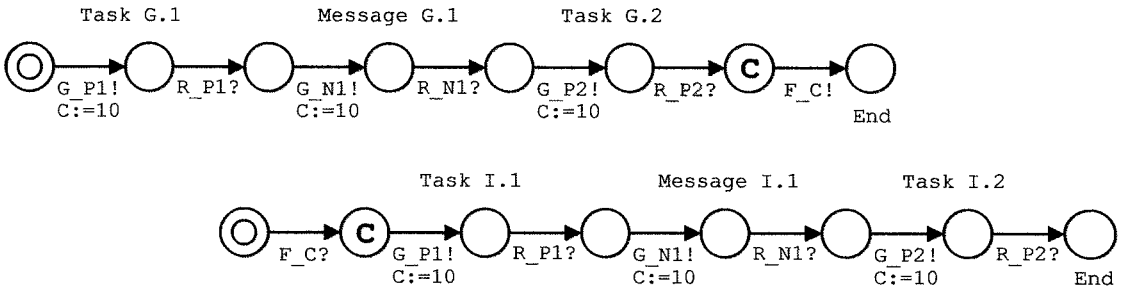


Figure 8.21: Timed Automata Model for Case ②

8.3.3 Precedence Constraint with Offset Time – Case ③

Finally, a precedence constraint which enables to give an execution order with offset between jobs is expected such as to allow a desire time to complete their works. It is

assumed that there are two jobs, Job G and Job J, in a system. This precedence constraint can enforce that the Job G starts first and then after the completion of the Job G, the Job J will start later but at a specific time, called the *offset*. This precedence constraint can be defined by using committed locations and an additional location represented the offset. Also, the F_C channel and an extra clock variable, named E_C, are required to use as a flag between the jobs and to measure the offset time respectively. Over the timed automata models for the Job G and Job J generated from the approach, a committed location is added to the end of the timed automata model for the Job G (after the Tasks G.2) and an extra location and committed location are added to the beginning of the time automata model for the Job J (before the Task J.1) with the channel and clock variable. So, having a synchronisation via the channel from the Job G, the Job J will wait at first as much as the offset, followed by its own work immediately after. By employing the committed locations, there will be no time delay between the jobs and the offset. Figure 8.22 includes the timed automata model for the precedence constraint with offset.

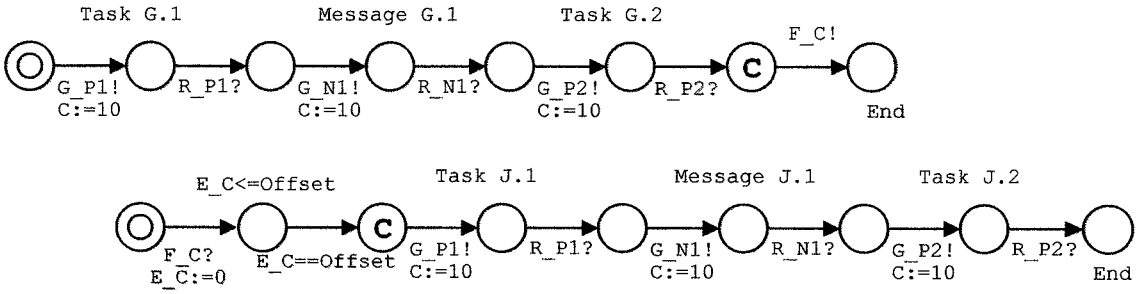


Figure 8.22: Timed Automata Model for Case ③

8.3.4 Experiments with Precedence Constraints

The timed automata models introduced for the precedence constraints between jobs are examined in the following experiments with the Robot Transport system. Not all the jobs from the system but some of them are extracted for the experiments and they are: CP-LR Command Job and LR Report Job working on between the Control Panel and Load Robot, and CP-UR Command Job and UR Report Job working on between the Control Panel and Unload Robot. It is assumed that these jobs require precedence constraints between the CP-LR Command Job and LR Report Job, and between the CP-

UR Command Job and UR Report Job, such that the reporting jobs always have to complete first as being referred by the commanding jobs. For example, the CP-LR Command Job works first, followed by the LR Report Job after the CP-LR Command Job completes.

When given the jobs with 100 time-units for the periods, 10 time-units for computation times and for the message transfer times, the approach originally generates a schedule shown in Figure 8.23: a blue colour presenting the CP-LR Command Job; a yellow colour for the LR Report Job; a green colour for the CP-UR Command Job; a tan colour for the UR Report Job. According to the schedule, it is understood that there are no precedence constraints between the jobs because they are occurred at random such as the CP-LR Command Job and LR Report Job start simultaneously, and also the CP-UR Command Job starts shortly before the UR Report Job meets its completion. Obviously, this is not an acceptable schedule for the system. Against this schedule, the precedence constraints described for the Case ② and Case ③ particularly are considered in the system (the precedence constraint in the Case ① is excluded as it is much simpler and easier to embody the constraint than the others).

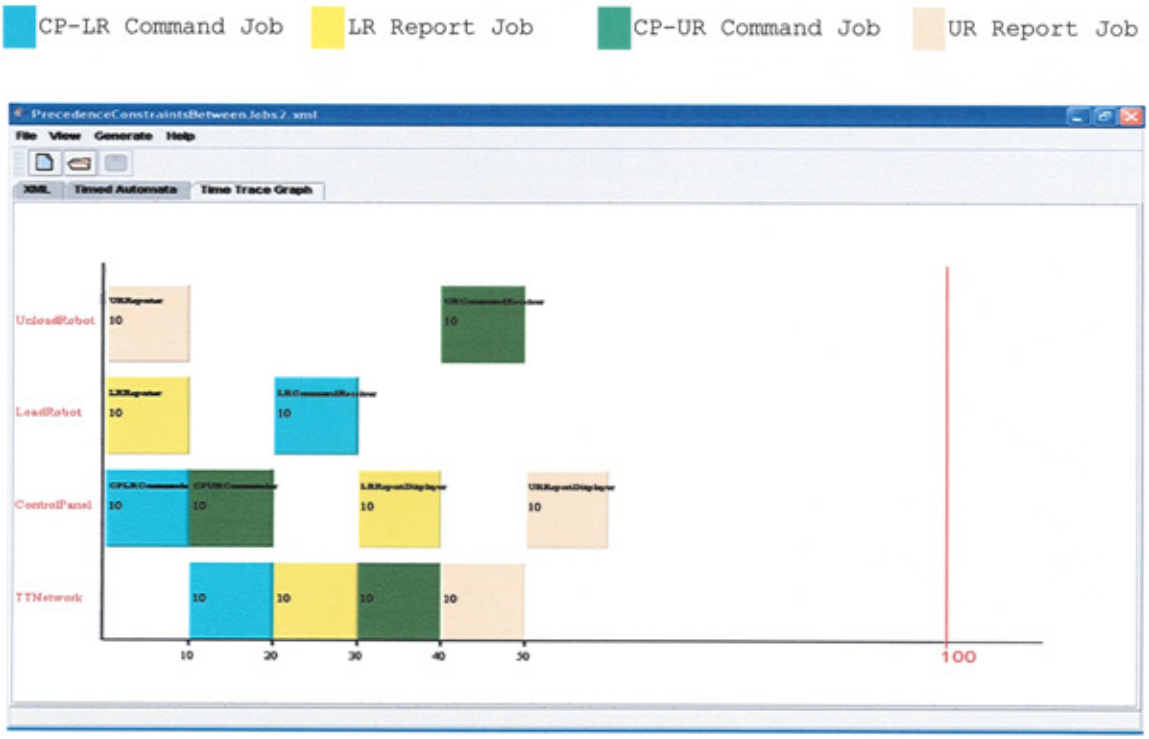


Figure 8.23: The Schedule with No Precedence Constraints.

The precedence constraint in the Case ② forces an execution order without time delay between jobs. And so, if this precedence constraint is applied to the system, the schedule generated from the approach is expected that there is an execution order between the jobs and also there are no time delay between the `CP-LR Command Job` and `LR Report Job` and also between the `CP-UR Command Job` and `UR Report Job`. When manually applying this, as the toolset has not been expanded yet to include this functionality, the approach generates a schedule shown in Figure 8.24. According to this schedule, it is found that there are precedence constraints between the jobs, such as once the `CP-LR Command Job` finishes its work at 30 time-units, the `LR Report Job` starts at the same time without time delay; likewise the `CP-UR Command Job` ends its works at 40 time-units and the `UR Report Job` starts at 40 time-units. Consequently, this schedule has the precedence constraints between the `CP-LR Command Job` and `LR Report Job`, and also between the `CP-UR Command Job` and `UR Report Job`. Also, there is no time delay between the jobs.

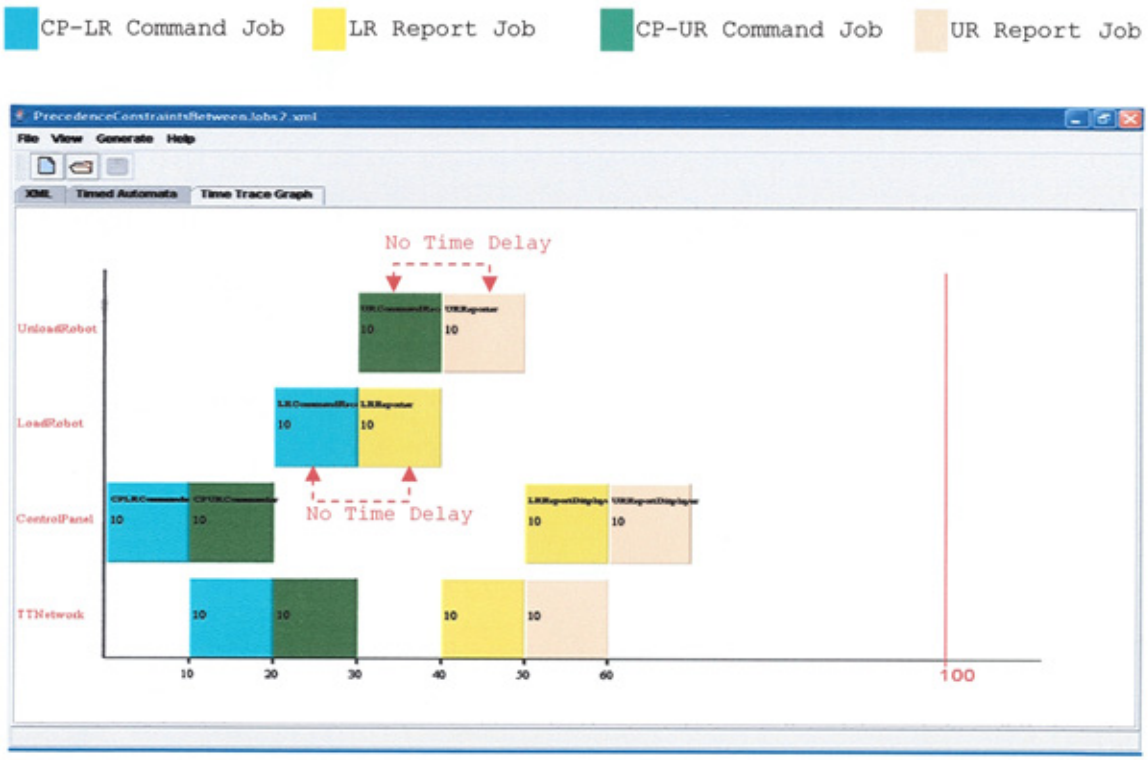


Figure 8.24: The Schedule for the Case ②

The precedence constraint in the Case ③ forces an execution order with an offset between jobs. Using this precedence constraint, the generated schedule for the system is expected that it has an execution order between the jobs and also there are offsets between the `CP-LR Command Job` and `LR Report Job` and also between the `CP-UR Command Job` and `UR Report Job`. When given the offset 15 time-units between the `CP-LR Command Job` and `LR Report Job`, and 25 time-units between the `CP-UR Command Job` and `UR Report Job`, the schedule generated manually from the approach is shown in Figure 8.25. According to the schedule, the `CP-LR Command Job` finishes at 30 time-units and the `LR Report Job` starts at 55 time-units; the `CP-UR Command Job` starts at 10 time-units and ends at 40 time-units, and the `UR Report Job` starts at 55 time-units. This means that there is an interval of 15 time-units, between the `CP-LR Command Job` and `LR Report Job` and also 25 time-units between the `CP-UR Command Job` and `UR Report Job`. These intervals are the same as the offsets imposed to the schedule. Thus, this schedule is acceptable for the system as it has the precedence constraints with offset between the jobs.

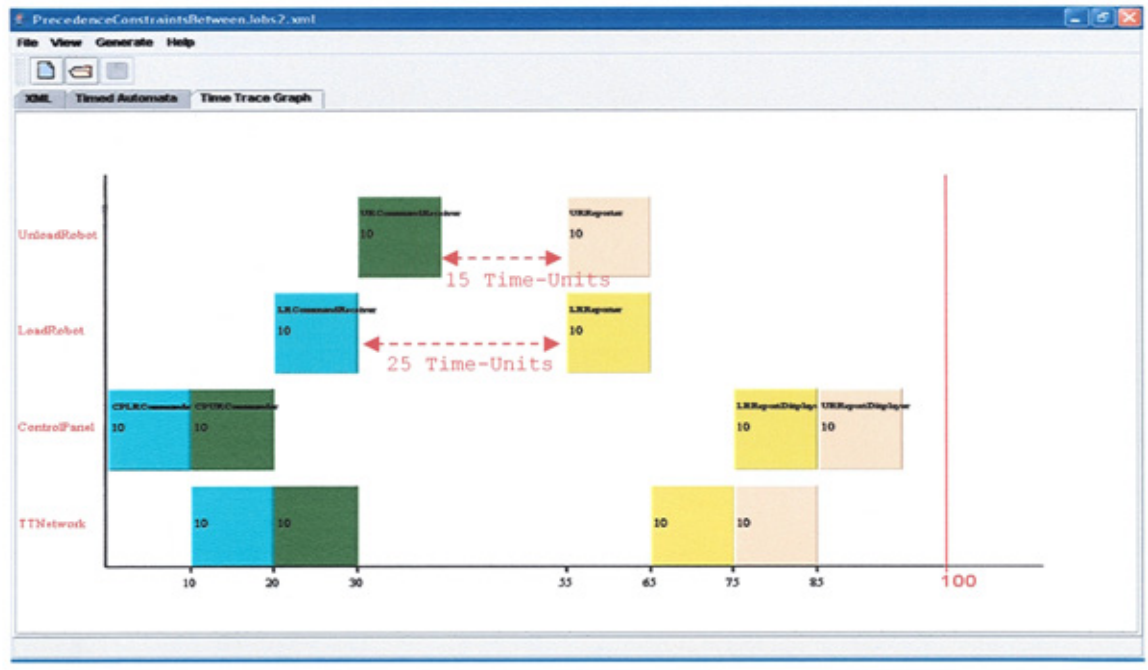


Figure 8.25: The Schedule for the Case ③

With the timed automata models introduced above, it is possible to impose precedence constraints between jobs and generate a schedule with these constraints. If there is a precedence constraint between jobs, the schedule can enable to prevent missing the latest information between them. Thus, it gives fast reaction in systems. However, there is a limitation in the timed automata models. As the models are based on the jobs which have the same period, they may not properly work if the jobs have the different period. In this case, some changes are further required in the models; nevertheless, the changes can be small as the fundamental principle of changing the models is not different.

8.4 Summary

Expanding the proposed approach, it is possible to yield schedules which can fulfil further schedule constraints for systems such as schedules with minimum jitter, schedules with compaction, etc. This chapter has explored such schedule constraints and introduced ways to simply impose the constraints using timed automata models and additional properties based on the approach.

- **Jitter Control.** Many real-time systems, in particular, in safety-critical areas required jitter to be strictly limited. With the Fluid Control system, two methods have been examined to synthesise a schedule considering no jitter, $S_{\overline{jitter}}$, based on the proposed approach: one of the methods uses a modified temporal property and the other uses modified UPPAAL models. The successful results from the methods have been presented and compared. According to the results, it has been recognised that using the models to generate a schedule with no jitter has less state explosion than using the property. However, there is more flexibility when using the property approach.
- **Scheduling Compaction.** In real-time systems using a time-triggered architecture, it is difficult to extend systems which are based on the pre-determined points in time and required a static schedule. This extensibility problem can be overcome by providing sufficient timing space in a scheduling round. Thus, based on the approach, the scheduling compaction has been introduced. Ideally, it is to find the compacted schedule in a scheduling round

by as much as pushing the approach until it is failed; the last successful schedule is adopted. The `System F` has been introduced as an experimental study and the necessity of a compacted schedule has been explained through the system, in order to reduce overhead of future expansion in a time-triggered architecture.

- **Precedence Constraints between Jobs.** Real-time systems may require precedence constraints between jobs for many reasons such as improving reaction between jobs. However, the timed automata model generated from the approach does not have a functionality to give the precedence constraints. Thus, three cases of the precedence constraints between jobs have been considered: a precedence constraint with any time, no time delay and offset. And the timed automata models for them have been introduced. When manually applying the models into the approach, the successful results have been introduced and illustrated.

Chapter 9

Conclusion and Future Work

This research has focused on the design of schedules for time-triggered architectures, in particular, for distributed real-time systems. With timed automata models, this thesis introduces a more rigorous approach to the automatic generation of schedules for such systems. This chapter summarises the contributions made by the thesis and suggests future avenues of research work.

9.1 Summary of the Work

With increasing functional demands in industry such as reliability, safety, flexibility, extensibility and testability, real-time systems are becoming crucial in a wide variety of fields in providing effective functional capabilities. Various types of real-time systems are distinguished by the demands: soft and hard, event and time-triggered, static and dynamic and off-line and on-line. In particular, the time-triggered architecture is becoming accepted as a means of implementing scalable, safer and more reliable solutions for distributed real-time systems.

In such systems, all activities take place in strictly fixed patterns and are scheduled in advance within a given scheduling round, even though they are distributed in the systems, and thus the activities can be executed in a more determined and reliable manner. However, this is the main obstacle in the design of time triggered systems because it is difficult to find a static schedule satisfying the constraints of all the activities within the scheduling round. The scheduler has to consider not only the requirements on each processor but also the global requirements of system-wide

behaviour including messages transmitted on networks. It is a major challenge to find an efficient way of building an appropriate schedule and generating it automatically.

This thesis has proposed a novel approach to designing schedules for distributed real-time systems using a time-triggered architecture, which is radically different from the existing mathematical methods or algorithms for schedule generation. The approach employs timed automata to model systems and verifies them with the UPPAAL verification engine.

9.1.1 Modelling the System

The distributed real-time system is modelled as a composition of the behaviours of a time-triggered architecture and the specific requirements of the system being designed. A system is a set of jobs, each of which is a sequence of tasks and messages. These finite sequences of tasks start when their inputs are available and finish executing by producing output values.

Formal definitions of terms used to construct distributed real-time systems using a time-triggered architecture have been established to ameliorate the confusion of terms differently referred to in the literature (Chapter 4). These terms, such as *system*, *processor*, *network*, *task*, *message* and *job*, provide unambiguous meaning for the formal expression of the systems, and further enable readers to correctly understand this thesis.

Based on the principle that a task and message can be modelled by finite states and a clock variable, a job consisting of tasks and messages is formalised using timed automata (Chapter 4). This exceptionally offers a way of modelling the system behaviour, which includes a number of jobs, in a semantically rigorous form.

UPPAAL, a model checking tool, was used to construct and analyse time automata models of systems (Chapter 5). In particular, three different UPPAAL automata model designs for a job have been proposed and examined to find a simple and efficient model for verification and schedule generation. This has demonstrated that the number of clock variables used in the models has more influence on verification times than other factors such as the number of locations and guards in the models. It is therefore important to minimise the number of clocks variables when creating a timed automata model.

9.1.2 Schedule Generation

Schedules are generated by a process of model checking. The novel approach is that timed automata models for a system can be verified with respect to a temporal logic property representing the system's specification. If the models satisfy the property, the UPPAAL verifier produces a diagnostic trace. This trace is analysed to automatically generate a feasible schedule for the system. In particular, the verification is based on an exhaustive state-space search, i.e. it examines every possible state of the models to verify the property. Therefore, the approach has the following characteristics:

- If a schedule exists for a system, the approach must find the schedule.
- If a schedule is not found by the approach, there is no schedule for a system. Thus, if a schedule does not exist for a system, the approach can identify its non-existence.
- If a schedule is identified by the approach, the schedule must be correct.

9.1.3 The Prototype Toolset

The modelling approach developed in this work is successful in schedule generation but there are difficulties in applying it manually because of the following reasons:

- **Creation**: Creating the models or including additional constraints requires considerable effort and care.
- **Analysis**: When analysing the trace file which includes substantial transition and state information, it is difficult to abstract data suiting our own purpose from the file.
- **Visualisation and Extraction**: It is difficult to visualise the form of schedules when just recorded as text files. Furthermore, it is tedious and error-prone to extract schedules from text files.

A prototype toolset has been developed to address these difficulties (Chapter 6). It provides a convenient, effective and standard way to apply the approach. The toolset has two components: Preprocessor and Analyser Engine. The Preprocessor provides a standard way of describing a system using XML which includes the behaviours and constraints for processors, networks, tasks, messages and jobs. This allows the

convenient addition to or change of a system's model. The Analyser Engine supports automatic schedule generation:

- Automatic creation of the timed automata models for a system and a temporal logic property for the system specification based on the XML.
- Full verification of the models and the property via the UPPAAL verifier.
- Analysis of the trace file to compute a schedule.
- Effective visualisation and extraction of a system schedule.

Using a simple example system, the application of the approach has been demonstrated using the prototype toolset to generate a successful schedule.

9.1.4 The Case Studies

Two case studies have been used to show the applicability and usability of the proposed approach using the prototype toolset (Chapter 7): the Adaptive Cruise Control system and the Robot Transport system. These case studies are of systems typical of those found in automobile and manufacturing industry. The first case study with the Adaptive Cruise Control system focused on illustrating each step of analysing the system using the prototype toolset and the successful generation of a schedule demonstrated that:

- If a schedule exists for a system, the approach must find the schedule.
- If a schedule is identified by the approach, the schedule must be correct.

In the second case study of the Robot Transport system, the effects of the approach varying timing parameters have been explored. It shows that:

- Arbitrary time-units assigned to a system do not have any influence on generating schedules. The approach is capable of adapting any arbitrary time-units.
- The greater the number of appearances of a job in a scheduling round, the longer it takes to find a schedule. In particular, this behaviour is strongly non linear as the number of jobs is increased.
- In general, systems with higher utilisation on processors and networks demand more schedule generation time than systems with lower utilisation. However, this is not always guaranteed when systems have tasks and messages allocated with different timing values individually. In this case, systems with lower utilisation demand more time to generate schedule than systems with higher

utilisation. Thus, assigning different timing values to each task and message in systems has more influence than assigning higher utilisation.

9.1.5 Constraining the Schedule

Having established the effectiveness of the approach, the technique has been successfully extended to yield schedules that embody further constraints on the systems (Chapter 8). In particular, three schedule constraints have been considered to demonstrate the flexibility of the approach.

- **Jitter Control.** To generate schedules with no jitter, two methods have been examined: one of the methods employs a modified temporal property, the other adopts a modified UPPAAL model. Applied to the Fluid Control system, which has a jitter problem, the methods successfully generated a new schedule with no jitter. Additionally, the result shows that the method using modified models is the more effective.
- **Scheduling Compaction.** To overcome the extensibility problem in a time-triggered architecture, a scheduling compaction method has been introduced. This method can effectively provide a compacted schedule and thus secure further scheduling freedom for future extension. With one of the case studies, the method has provided compacted schedules for the system and demonstrated how to deploy the additional scheduling space to schedule additional jobs.
- **Precedence Constraints between Jobs.** To introduce precedence constraints between jobs, three types of precedence constraints have been considered: a precedence constraint between jobs with any time, no time delay and with offsets. These constraints provide a way of enforcing a fast reaction between jobs. Consequently, three examples of timed automata models have employed these constraints. With part of the Robot Transport system, each of these timed automata models has been demonstrated by computing schedules satisfying the additional constraints.

9.2 Future Work

To further enhance the approach and toolset that have been so far developed, the following work is proposed.

9.2.1 The Toolset

The toolset could benefit from a number of improvements:

- **IDE Interface.** The direct entry of XML is tedious and error-prone. A graphical development environment could be provided to automatically generate and check the required XML output.
- **Various Timed Automata Model Generation.** Currently the toolset does not enable automatic application of further schedule constraints such as jitter problem, compression, precedence constraints as described in Chapter 8. An extension to en-corporate these features would increase the applicability of the toolset.
- **Forward Integration with Target Implementations.** Usability would be improved if the toolset automatically generated task and message schedules compatible with target software.
- **Backward Integration with System Specification tools.** The method builds on system task/message timing and dependency specification. Some form of integration with those tools/methods that elicit this information is recommended.

9.2.2 Practical Case Studies

Although this thesis has included three representative case studies, these cases are necessarily limited and further studies are warranted:

- **Experimental Case Studies.** The cases studies applied in this thesis briefly have examined the efficiency and range of the approach. To further explore the efficiency of the approach, additional case studies are needed. Of particular interest is the state-space explosion problem. Greater knowledge of the problem could be used to develop design guidelines.
- **Real Case Studies.** This thesis has focused on generating schedules. To more fully demonstrate the applicability of the approach, a full-scale industrial

validation is recommended. This should involve the full design cycle. In particular, the implementation of generated schedule on real target software and hardware will be examined. This will bring the methods to bear on real tasks and processors, real message and networks, scheduler and operations systems, etc. This work will enable a further refinement of the approach.

Appendix A

Mathematical Symbols

This appendix lists those mathematical symbols that are used in several places in the thesis.

Symbols for Timed Automata Models

$\mathcal{C}$	A set of clocks.
$\mathcal{C}^0$	A set of clocks and the value 0.
$cf(c)$	A function, mapping each clock $c \in \mathcal{C}$ to the largest integer constant.
$fr(d)$	The fractional part of d.
$\mathbb{R}^+$	Non-negative real-value.
$\mathcal{V}_C$	The set of valuations over of clocks $\mathcal{C}$.
$v(x)$	The valuation of clock x where $v \in \mathcal{V}_C$.
$v \sim v'$	An equivalence relation between v and v' valuation.
$v \models \psi$	The satisfaction of clock constraints $\psi \in \Psi_C$ over clock $v \in \mathcal{V}_C$ valuation.
Γ_C	The set of assignments over clocks, $\mathcal{C}$.
$\gamma(x)$	The assignment of x to a clock where $\gamma \in \Gamma_C$.
Ψ_C	The set of clock constraints over $\mathcal{C}$.
ψ	A clock constraint, $\psi \in \Psi_C$
$\mathcal{S}$	A finite set of locations.
$\mathcal{L}$	A finite set of labels (or actions).
$\mathcal{I}$	A finite set of mapping from locations to clock constraints.

$\mathcal{I}(s)$	The invariant of $s \in \mathcal{S}$.
$\mathcal{E}$	A finite set of edges defined by the tuple (s,l,ψ,γ,s').
$\mathcal{A}$	A timed automaton defined by the tuple $(\mathcal{S},\mathcal{C},\mathcal{L},\mathcal{I},\mathcal{E})$.
$\mathcal{A}_i \parallel \mathcal{A}_j$	The parallel composition of timed Automaton $\mathcal{A}_i$ and $\mathcal{A}_j$
$\mathcal{Q}$	A set of states.
$\rightarrow$	A set of transitions where $\rightarrow \subseteq \mathcal{Q} \times \mathcal{L} \times \mathcal{Q}$.
$\mathcal{R}e(\mathcal{A})$	A region automaton.
$[v]$	The set of clock assignments over clock regions.
$\mathcal{Z}o(\mathcal{A})$	A zone automaton.
z	A clock zone.

Symbols for System Models

T_i	The period of task *i*.
T_{P_i}	The set of tasks on the processor P_i or simply T_i.
t_i	A task *i* where $t_i \in T$.
p_i	The periods of task t_i.
R_i	The worst case response time of task *i*.
$hp(i)$	The set of tasks higher in priority than the task t_i.
C_i	The computation time of task t_i.
$c_i^{\uparrow}$	The maximum computation time of task t_i.
$c_i^{\downarrow}$	The minimum computation time of task t_i.
D_i	The deadline of task t_i.
$\mathcal{J}$	A finite set of Jobs.
J	A job where $J \in \mathcal{J}$
$d^{\uparrow}$	A maximum deadline of J.
$d^{\downarrow}$	A minimum deadline of J.
w_f	The finish task of a job.

w_i	Either a task or message where $w_i \in W$.
w_0	The start task of a job.
I_i	The interference of task *i* from higher-priority tasks.
M_{N_i}	The set of messages on the network N_i or simply M_i.
m_i	A message *i* where $m_i \in M$.
π_i	The transfer time of m_i.
MB	The set of mailbox for tasks and messages.
mb_i	The mailbox of t_i where $mb_i \in MB$.
$\overrightarrow{mb}$	An input of a mailbox.
$\underrightarrow{mb}$	An output of a mailbox.
$\mathcal{N}$	A finite set of networks.
N	A network where $N \in \mathcal{N}$.
$\mathcal{P}$	A finite set of processors.
P	A processor where $P \in \mathcal{P}$.
$\mathcal{R}$	The set of processor and networks $\mathcal{P} \cup \mathcal{N}$.
S	A system consisting of processors, networks and jobs. $\mathcal{S}=(\mathcal{P},\mathcal{N},\mathcal{J})$.
S_{ch}	A feasible schedule.
$S_{\overline{jitter}}$	A schedule with no jitter.
W	The set $T \cup M$ of both tasks and messages.
$q_{\overline{w}}$	The initial state of w.
q_w	The active state of w.
$q_{\underline{w}}$	The finial state of w.
$finish(x)$	A function which indicates the finish time of x.
$Jitter(x)$	The jitter of x.
$n(x)$	A function indicating list set of x.
$start(x)$	A function which indicates the start time of x.

Symbols for Temporal Logic Properties

G	A temporal operator, meaning 'Always'.
F	A temporal operator, meaning 'Eventually'.
A	A temporal operator, meaning 'All'.
E	A temporal operator, meaning 'Some'.
AF	A combination of temporal operators, meaning 'Eventually all'.
AG	A combination of temporal operators, meaning 'Always all'.
EF	A combination of temporal operators, meaning 'Eventually some'.
EG	A combination of temporal operators, meaning 'Always some'.

Appendix B

XML Description for Systems

This appendix includes the XML description for the systems which are used in the thesis: Fluid Control System (Chapter 6 and Chapter 8), Adapted Cruise Control System (Chapter 7) and Robot Transport System (Chapter 7).

B.1 Fluid Control System

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="tt_system.css"?>
<!DOCTYPE tt_system SYSTEM "tt_system.dtd">

<tt_system>
   <systemID>FluidControlSystem</systemID>
   <processor>
      <processorID> plant </processorID>
      <task>
         <taskID> sample </taskID>
         <period> 100 </period>
         <computation> 10 </computation>
         <pc_in> environment </pc_in>
         <pc_out> pressure </pc_out>
      </task>
      <task>
         <taskID> actuator </taskID>
         <period> 100 </period>
         <computation> 10 </computation>
         <pc_in> valve </pc_in>
         <pc_out> environment </pc_out>
      </task>
      <task>
         <taskID> alarm </taskID>
         <period> 50 </period>
         <computation> 10 </computation>
         <pc_in> environment </pc_in>
         <pc_out> alarm </pc_out>
      </task>
   </processor>
   <processor>
      <processorID> consol </processorID>
```

```
      <task>
         <taskID> controller </taskID>
         <period> 100 </period>
         <computation> 10 </computation>
         <pc_in> pressure </pc_in>
         <pc_out> valve </pc_out>
      </task>
      <task>
         <taskID> indicator </taskID>
         <period> 50 </period>
         <computation> 10 </computation>
         <pc_in> alarm </pc_in>
         <pc_out> environment </pc_out>
      </task>
   </processor>

   <network>
      <networkID>ttp</networkID>
      <message>
         <messageID> pressure </messageID>
         <period> 100 </period>
         <transfer_time> 10 </transfer_time>
         <pc_in> sample </pc_in>
         <pc_out> controller </pc_out>
      </message>
      <message>
         <messageID> valve </messageID>
         <period> 100 </period>
         <transfer_time> 10 </transfer_time>
         <pc_in> controller </pc_in>
         <pc_out> actuator </pc_out>
      </message>
      <message>
         <messageID> alarm </messageID>
         <period> 50 </period>
         <transfer_time> 10 </transfer_time>
         <pc_in> alarm </pc_in>
         <pc_out> indicator </pc_out>
      </message>
   </network>

   <job>
      <jobID> control_loop </jobID>
      <from>
         <when> start </when>
         <taskID> sample </taskID>
         <processorID> plant </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> actuator </taskID>
         <processorID> plant </processorID>
      </to>
      <within> 100 </within>
   </job>
   <job>
      <jobID> alarm </jobID>
```

```
      <from>
         <when> start </when>
         <taskID> alarm </taskID>
         <processorID> plant </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> indicator </taskID>
         <processorID> consol </processorID>
      </to>
      <within> 50 </within>
   </job>
</tt_system>
```

B.2 Adaptive Cruise Control System

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="tt_system.css"?>
<!DOCTYPE tt_system SYSTEM "tt_system.dtd">

<tt_system>
   <systemID>AdaptiveCriuseControlSystem</systemID>
   <processor>
      <processorID> EngineControlUnit </processorID>
      <task>
         <taskID> EngineActuator </taskID>
         <period> 50 </period>
         <computation> 10 </computation>
         <pc_in> EngineMessage </pc_in>
         <pc_out> Environment </pc_out>
      </task>
   </processor>
   <processor>
      <processorID> ConsoleControlUnit </processorID>
      <task>
         <taskID> CruiseIndicator </taskID>
         <period> 100 </period>
         <computation> 10 </computation>
         <pc_in> Environment </pc_in>
         <pc_out> IndicatorMessage </pc_out>
      </task>
      <task>
         <taskID> ConsoleReporter </taskID>
         <period> 200 </period>
         <computation> 10 </computation>
         <pc_in> ConsoleReporter </pc_in>
         <pc_out> Environment </pc_out>
      </task>
   </processor>
   <processor>
      <processorID> BrakeControlUnit </processorID>
      <task>
         <taskID> RevolutionReader  </taskID>
         <period> 200 </period>
```

```
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> RevolutionMessage </pc_out>
    </task>
    <task>
        <taskID> BrakeActuator  </taskID>
        <period> 50 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> BrakeMessage </pc_out>
    </task>
 </processor>
<processor>
    <processorID> AdaptiveCruiseUnit </processorID>
    <task>
        <taskID> ConsoleReader </taskID>
        <period> 100 </period>
        <computation> 10 </computation>
        <pc_in> IndicatorMessage </pc_in>
        <pc_out> Environment </pc_out>
    </task>
    <task>
        <taskID> EngineSet </taskID>
        <period> 50 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> EngineMessage </pc_out>
    </task>
    <task>
        <taskID> BrakeSet </taskID>
        <period> 50 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> BrakeMessage </pc_out>
    </task>
    <task>
        <taskID> WheeelReader </taskID>
        <period> 200 </period>
        <computation> 10 </computation>
        <pc_in> RevolutionMessage </pc_in>
        <pc_out> WheelMessage </pc_out>
    </task>
 </processor>

 <network>
    <networkID>ttp</networkID>
    <message>
        <messageID> EngineMessage </messageID>
        <period> 50 </period>
        <transfer_time> 10 </transfer_time>
        <pc_in> EngineSet </pc_in>
        <pc_out> EngineActuator </pc_out>
    </message>
    <message>
        <messageID> BrakeMessage </messageID>
        <period> 50 </period>
        <transfer_time> 10 </transfer_time>
```

```
        <pc_in> BrakeSet </pc_in>
        <pc_out> BrakeActuator </pc_out>
    </message>
    <message>
        <messageID> RevolutionMessage </messageID>
        <period> 200 </period>
        <transfer_time> 10 </transfer_time>
        <pc_in> RevolutionReader </pc_in>
        <pc_out> WheelReader </pc_out>
    </message>
    <message>
        <messageID> WheelMessage </messageID>
        <period> 200 </period>
        <transfer_time> 10 </transfer_time>
        <pc_in> WheelReader </pc_in>
        <pc_out> ConsoleReporter </pc_out>
    </message>
    <message>
        <messageID> IndicatorMessage </messageID>
        <period> 100 </period>
        <transfer_time> 10 </transfer_time>
        <pc_in> CruiseIndicator </pc_in>
        <pc_out> ConsoleReader </pc_out>
    </message>
</network>

<job>
    <jobID> ConsoleCruiseJob </jobID>
    <from>
        <when> start </when>
        <taskID> CruiseIndicator </taskID>
        <processorID> ConsoleControlUnit </processorID>
    </from>
    <to>
        <when> end </when>
        <taskID> ConsoleReader </taskID>
        <processorID> AdaptiveCruiseUnit </processorID>
    </to>
    <within> 100 </within>
</job>
<job>
    <jobID> SpeedCruiseJob </jobID>
    <from>
        <when> start </when>
        <taskID> EngineSet </taskID>
        <processorID> AdaptiveCruiseUnit </processorID>
    </from>
    <to>
        <when> end </when>
        <taskID> EngineActuator </taskID>
        <processorID> EngineControlUnit </processorID>
    </to>
    <within> 50 </within>
</job>
 <job>
    <jobID> BrakeCruiseJob </jobID>
    <from>
```

```
            <when> start </when>
            <taskID> BrakeSet </taskID>
            <processorID> AdaptiveCruiseUnit </processorID>
         </from>
         <to>
            <when> end </when>
            <taskID> BrakeActuator </taskID>
            <processorID> BrakeControlUnit </processorID>
         </to>
         <within> 50 </within>
      </job>
      <job>
         <jobID> ConsoleReportJob </jobID>
         <from>
            <when> start </when>
            <taskID> RevolutionReader </taskID>
            <processorID> BrakeControlUnit </processorID>
         </from>
         <to>
            <when> end </when>
            <taskID> ConsoleReporter </taskID>
            <processorID> ConsoleControlUnitsol </processorID>
         </to>
         <within> 100 </within>
      </job>
</tt_system>
```

B.3 Robot Transport System

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="tt_system.css"?>
<!DOCTYPE tt_system SYSTEM "tt_system.dtd">

<tt_system>
   <systemID>RobotTransportSystem</systemID>
   <processor>
      <processorID> ControlPanel </processorID>
      <task>
         <taskID> CP-LRCommander </taskID>
         <period> 200 </period>
         <computation> 10 </computation>
         <pc_in> Environment </pc_in>
         <pc_out> CP-LRCommandMessage </pc_out>
      </task>
      <task>
         <taskID> LRReportDisplayer </taskID>
         <period> 100 </period>
         <computation> 10 </computation>
         <pc_in>  LRReportMessage </pc_in>
         <pc_out> Environment </pc_out>
      </task>
      <task>
         <taskID> CP-URCommander </taskID>
         <period> 200 </period>
```

```
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> CP-URCommandMessage </pc_out>
    </task>
    <task>
        <taskID> URReportDisplayer </taskID>
        <period> 100 </period>
        <computation> 10 </computation>
        <pc_in> URReportMessage </pc_in>
        <pc_out> Environment </pc_out>
    </task>
    <task>
        <taskID> ConveyorReportDisplayer </taskID>
        <period> 100 </period>
        <computation> 10 </computation>
        <pc_in> ConveyorReportMessage </pc_in>
        <pc_out> Environment </pc_out>
    </task>
</processor>
<processor>
    <processorID> Conveyor </processorID>
    <task>
        <taskID> CON-LRCommander </taskID>
        <period> 400 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> Con-LRCommandMessage </pc_out>
    </task>
    <task>
        <taskID> CON-URCommander </taskID>
        <period> 400 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> Con-URCommandMessage </pc_out>
    </task>
    <task>
        <taskID> ConveyorActuator </taskID>
        <period> 50 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> Environment </pc_out>
    </task>
    <task>
        <taskID> ConveyorReporter </taskID>
        <period> 100 </period>
        <computation> 10 </computation>
        <pc_in> Environment </pc_in>
        <pc_out> ConveyorReportMessage </pc_out>
    </task>
</processor>
<processor>
    <processorID> LoadRobot </processorID>
    <task>
        <taskID> LRCommandReceiver </taskID>
        <period> 200 </period>
        <computation> 10 </computation>
        <pc_in> CP-LRCommandMessage </pc_in>
```

```
            <pc_out> Environment </pc_out>
        </task>
        <task>
            <taskID> LRReporter </taskID>
            <period> 100 </period>
            <computation> 10 </computation>
            <pc_in> Environment </pc_in>
            <pc_out> LRReportMessage </pc_out>
        </task>
        <task>
            <taskID> LRConveyorCommandReceiver </taskID>
            <period> 400 </period>
            <computation> 10 </computation>
            <pc_in> Con-LRCommandMessage </pc_in>
            <pc_out> Environment </pc_out>
        </task>
        <task>
            <taskID> LRActuator </taskID>
            <period> 50 </period>
            <computation> 10 </computation>
            <pc_in> Environment </pc_in>
            <pc_out> Environment </pc_out>
        </task>
    </processor>
    <processor>
        <processorID> UnloadRobot </processorID>
        <task>
            <taskID> URCommandReceiver </taskID>
            <period> 200 </period>
            <computation> 10 </computation>
            <pc_in> CP-URCommandMessage </pc_in>
            <pc_out> Environment </pc_out>
        </task>
        <task>
            <taskID> URReporter </taskID>
            <period> 100 </period>
            <computation> 10 </computation>
            <pc_in> Environment </pc_in>
            <pc_out> URReportMessage </pc_out>
        </task>
        <task>
            <taskID> URConveyorCommandReceiver </taskID>
            <period> 400 </period>
            <computation> 10 </computation>
            <pc_in> Con-URCommandMessage </pc_in>
            <pc_out> Environment </pc_out>
        </task>
        <task>
            <taskID> URActuator </taskID>
            <period> 50 </period>
            <computation> 10 </computation>
            <pc_in> Environment </pc_in>
            <pc_out> Environment </pc_out>
        </task>
    </processor>

    <network>
```

```
    <networkID>TTNetwork</networkID>
    <message>
       <messageID> CP-LRCommandMessage </messageID>
       <period> 400 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> CP-LRCommander </pc_in>
       <pc_out> LRCommandReceiver </pc_out>
    </message>
    <message>
       <messageID> LRReportMessage </messageID>
       <period> 200 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> LRReporter </pc_in>
       <pc_out> LRReportDisplayer </pc_out>
    </message>
    <message>
       <messageID> Con-LRCommandMessage </messageID>
       <period> 400 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> Con-LRCommander </pc_in>
       <pc_out> LRConveyorCommandReceiver </pc_out>
    </message>
    <message>
       <messageID> CP-URCommandMessage </messageID>
       <period> 400 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> CP-URCommander </pc_in>
       <pc_out> URCommandReceiver </pc_out>
    </message>
    <message>
       <messageID> URReportMessage </messageID>
       <period> 200 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> URReporter </pc_in>
       <pc_out> URReportDisplayer </pc_out>
    </message>
    <message>
       <messageID> Con-URCommandMessage </messageID>
       <period> 400 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> Con-URCommander </pc_in>
       <pc_out> URConveyorCommandReceiver </pc_out>
    </message>
    <message>
       <messageID> ConveyorReportMessage </messageID>
       <period> 100 </period>
       <transfer_time> 10 </transfer_time>
       <pc_in> ConveyorReporter </pc_in>
       <pc_out> ConveyorReportDisplayer </pc_out>
    </message>
 </network>

 <job>
    <jobID> CP-LRCommandJob </jobID>
    <from>
       <when> start </when>
       <taskID> CP-LRCommander </taskID>
```

```
            <processorID> ControlPanel </processorID>
        </from>
        <to>
            <when> end </when>
            <taskID> LRCommandReceiver </taskID>
            <processorID> LoadRobot </processorID>
        </to>
        <within> 200 </within>
    </job>
    <job>
        <jobID> LRReportJob </jobID>
        <from>
            <when> start </when>
            <taskID> LRReporter </taskID>
            <processorID> LoadRobot </processorID>
        </from>
        <to>
            <when> end </when>
            <taskID> LRReportDisplayer </taskID>
            <processorID> ControlPanel </processorID>
        </to>
        <within> 100 </within>
    </job>
     <job>
        <jobID> Con_LRCommandJob </jobID>
        <from>
            <when> start </when>
            <taskID> CON-LRCommander </taskID>
            <processorID> Conveyor </processorID>
        </from>
        <to>
            <when> end </when>
            <taskID> LRConveyorCommandReceiver </taskID>
            <processorID> LoadRobot </processorID>
        </to>
        <within> 400 </within>
    </job>
    <job>
        <jobID> CP-URCommandJob </jobID>
        <from>
            <when> start </when>
            <taskID> CP-URCommander </taskID>
            <processorID> ControlPanel </processorID>
        </from>
        <to>
            <when> end </when>
            <taskID> URCommandReceiver </taskID>
            <processorID> UnloadRobot </processorID>
        </to>
        <within> 200 </within>
    </job>
    <job>
        <jobID> URReportJob </jobID>
        <from>
            <when> start </when>
            <taskID> URReporter </taskID>
            <processorID> UnloadRobot </processorID>
```

```
      </from>
      <to>
         <when> end </when>
         <taskID> URReportDisplayer </taskID>
         <processorID> ControlPanel </processorID>
      </to>
      <within> 100 </within>
   </job>
   <job>
      <jobID> Con-URCommandJob </jobID>
      <from>
         <when> start </when>
         <taskID> CON-URCommander </taskID>
         <processorID> Conveyor </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> URConveyorCommandReceiver </taskID>
         <processorID> UnlaodRobot </processorID>
      </to>
      <within> 400 </within>
   </job>
   <job>
      <jobID> ConveyorReportJob </jobID>
      <from>
         <when> start </when>
         <taskID> ConveyorReporter </taskID>
         <processorID> Conveyor </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> ConveyorReportDisplayer </taskID>
         <processorID> ControlPanel </processorID>
      </to>
      <within> 100 </within>
   </job>
   <job>
      <jobID> LRMoveJob </jobID>
      <from>
         <when> start </when>
         <taskID> LRActuator </taskID>
         <processorID> LoadRobot </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> LRActuator </taskID>
         <processorID> LoadRobot </processorID>
      </to>
      <within> 50 </within>
   </job>
   <job>
      <jobID> URMoveJob </jobID>
      <from>
         <when> start </when>
         <taskID> URActuator </taskID>
         <processorID> UnloadRobot </processorID>
      </from>
```

```
      <to>
         <when> end </when>
         <taskID> URActuator </taskID>
         <processorID> UnloadRobot </processorID>
      </to>
      <within> 50 </within>
   </job>
   <job>
      <jobID> ConveyorMoveJob </jobID>
      <from>
         <when> start </when>
         <taskID> ConveyorActuator </taskID>
         <processorID> Conveyor </processorID>
      </from>
      <to>
         <when> end </when>
         <taskID> ConveyorActuator </taskID>
         <processorID> Conveyor </processorID>
      </to>
      <within> 50 </within>
   </job>
</tt_system>
```

Appendix C

User Guide of the Prototype Toolset

This appendix describes the basic operation of the prototype toolset, taking account of how to load XML description for systems, how to examine such a description and finally how to generate timed automata models and a schedule graph. It is assumed that the toolset have been properly installed, in particular, with a java virtual machine.

C.1 Loading XML Description into the Tool

Figure C.1 shows the main window of the prototype toolset which basically includes three tabs named as XML, Timed Automata, Time Trace Graph. The XML tab is to load a XML description for systems which is desired to generate a schedule; the Timed Automata tab is to include the textual of timed automata models and a property generated by the toolset; the Time Trace Graph tab is to show a schedule graph. Click **File→New** in order to make a new description.

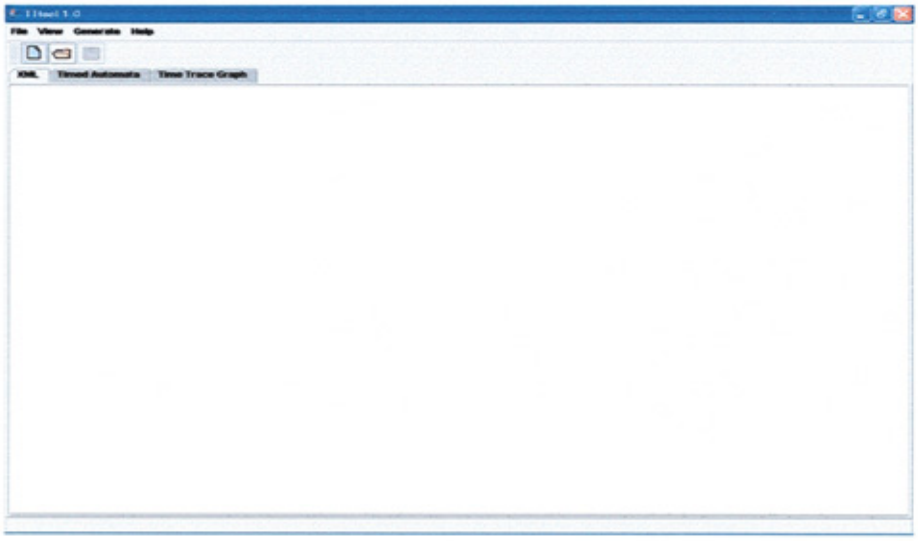


Figure C.1: Prototype Toolset Main Window

There are two ways to include such a description into the XML tab either by directly typing it or by indirectly loading a file already created using text-edit tools such as `NotePad` or `WordPad` windows applications. When loading the file, click **File→Open** menu or an icon for the loading, and then select it from a standard open dialog box, as shown in Figure C.2.

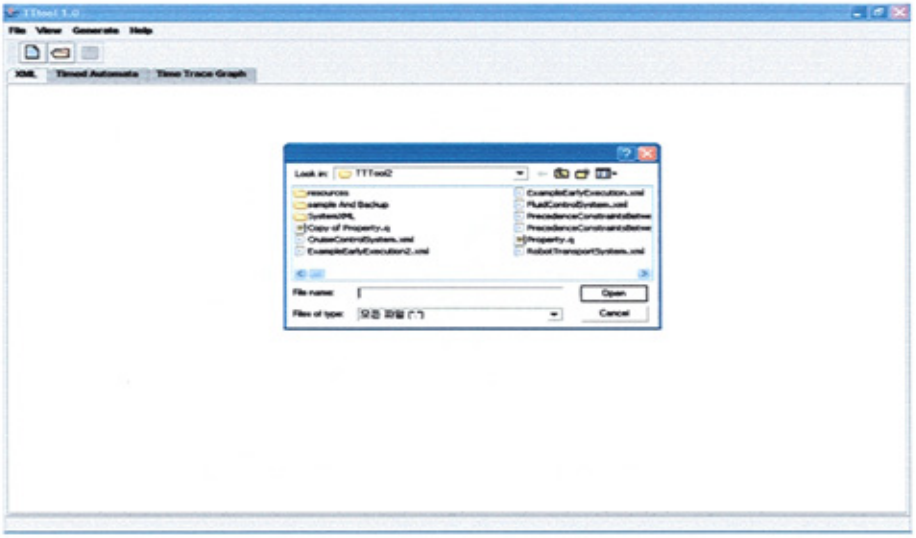


Figure C.2: Loading XML Description through a Open Dialog Box

The loading XML description will be shown in the tab after selecting it, like Figure C.3. There is still chances to amend the description via directly the tab. At the moment, the XML tab is so simple but will be expected to introduce an efficient IDE functionality in future.

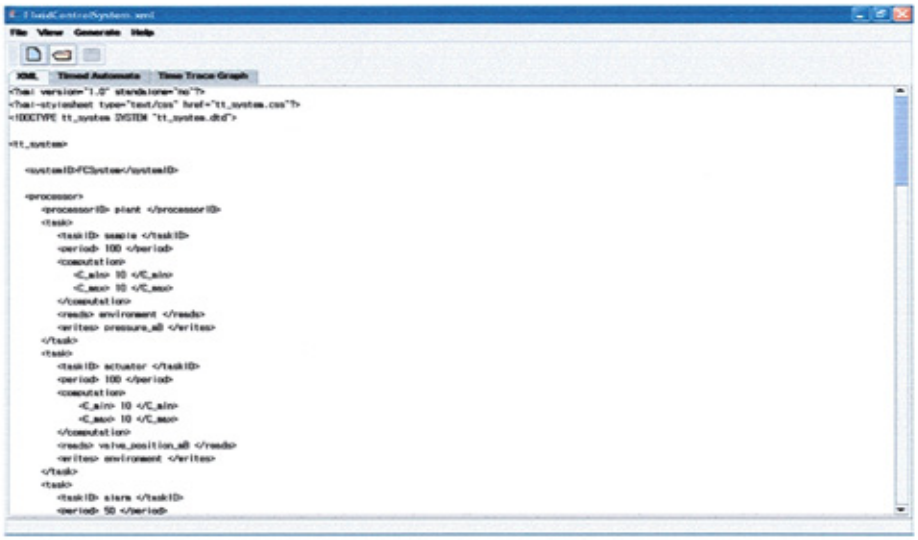


Figure C.3: XML Description in the XML Tab

C.2 Examining XML Description

After loading a XML Description, there are two ways of examining the description. First, click **View→XML Data Tree**; this shows all the data from the description as a tree-structured format, similar to Windows Explorer. Each folder icon in the tree represents a processor or network, etc; each text icon in the tree shows the value of its higher component. See Figure C.4.

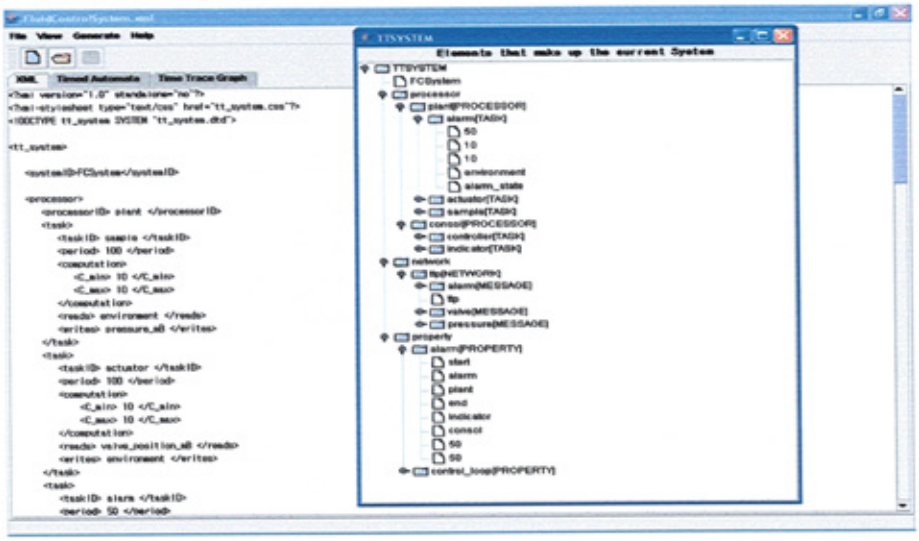


Figure C.4: XML Data Tree Window

Second, click **View→Job Mailboxes**; this examines the correctness of mailboxes set for jobs, showing all the mailboxes from the start task to the end task of a job. When the set is correct, the whole task and message links of jobs can be shown like Figure C.5.

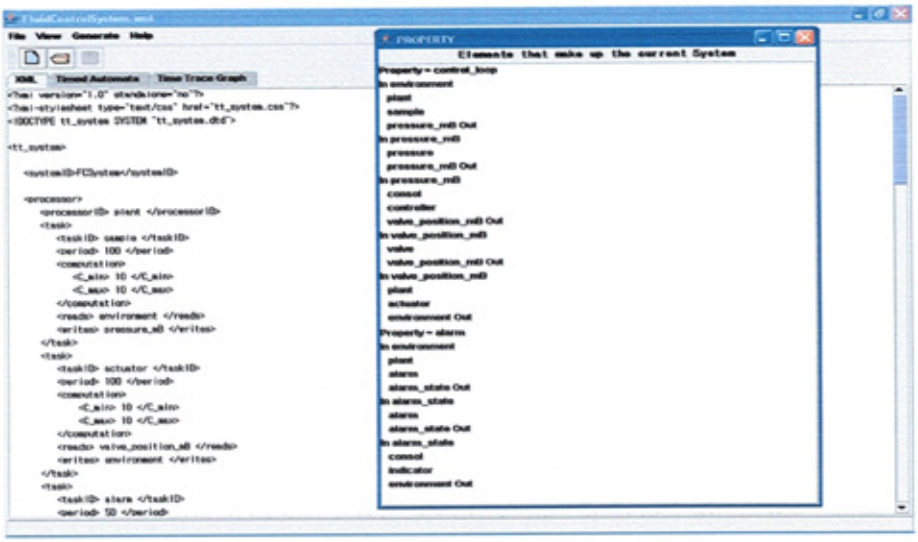


Figure C.5: XML Chain Tree Window

C.3 Generating Timed Automata Models and a Schedule Graph

To generate timed automata models and a property based on a XML description, click **Generate→Timed Automata** and then go to the Timed Automata tab. There are two text areas in the tab; the upper area includes the generated property and the lower area contains the textual format of timed automata models generated by the toolset.

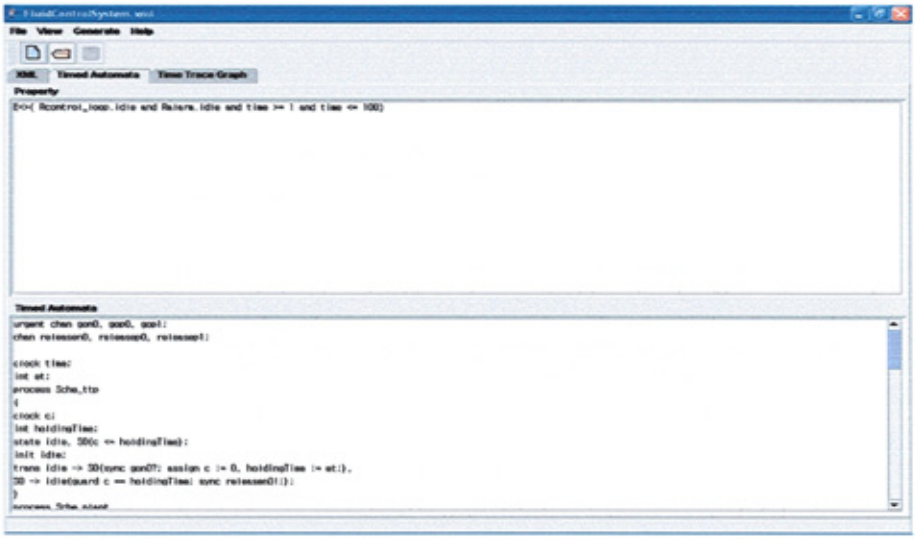


Figure C.6: Timed Automata Models and a Property in the Timed Automata Tab

To generate a schedule graph, click **Generate→Time Trace Graph**. This menu can be selected only after generating timed automata models. In the graph, the rows are labelled with the processor names or network name, and the columns indicate time values; each colour in the graph represents a job.

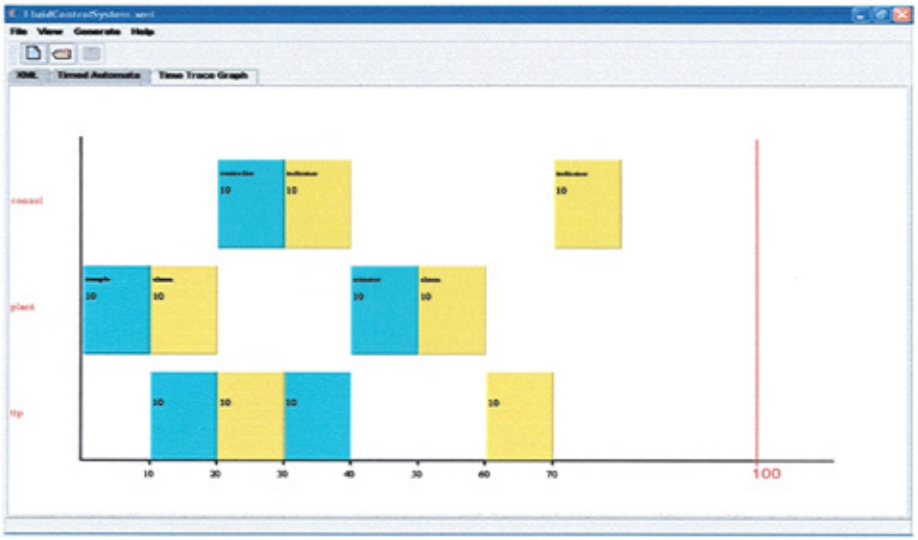
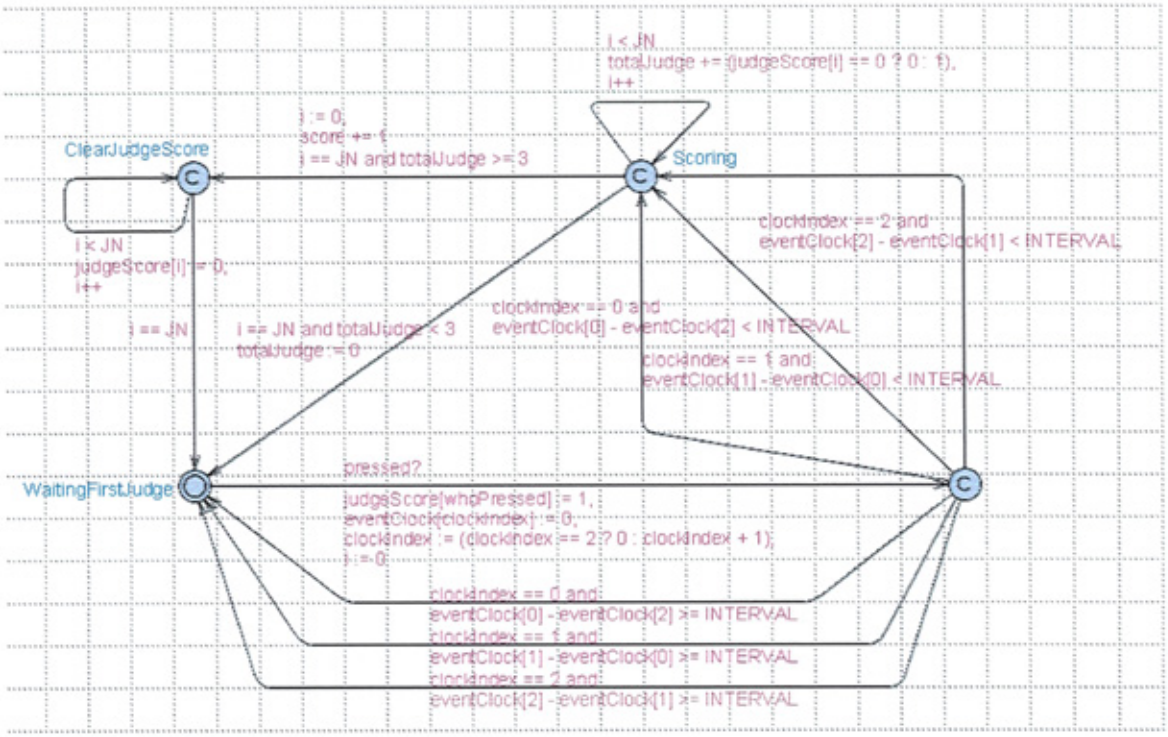


Figure C.7: Schedule Graph Generated by the Toolset

Appendix D

Enhanced Olympic Boxing Scoring System Model

This model is additional Score Controller model to fix the scheduling problem pointed by Alan's private correspondence.



Bibliography

[AB90] N. Audsley and A. Burns, Real-Time System Scheduling, University of York - Department of Computer Science Report YCS 134, 1990.

[ABL98] L. Aceto, A. Burgueño and K. G. Larsen. Model Checking via Reachability Testing for Timed Automata, In *Proc. of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 263-280, 1998.

[ABD+95] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell and A. J. Wellings, Fixed Priority Pre-emptive Scheduling: An Historical Perspective, Real-Time Systems, volume 8, issue 2-3, pages 173-198, 1995.

[ABR+91] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, Hard Real-Time Scheduling: The Deadline Monotonic Approach, In *the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA, 1991.

[ABR+93] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell and A. J. Wellings, Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling, *Software Engineering Journal*, volume 8, pages 284-292, 1993.

[ACC05] Adaptive Cruise Control System Overview, In *the 5th meeting of the US Software System Safety Working Group*, April 2005.

[ACD90] R. Alur, C. Courcoubetis and D. Dill, Model-Checking for Real-Time Systems. *Logic in Computer Science*, pages 414-425. IEEE Computer Society Press, June 1990.

[AD90] R. Alur and D. Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, pages 322-355, July 1990.

[AD94] R. Alur and D. Dill, A Theory of Timed Automata, *Theoretical Computer Science*, volume 126, no 2, pages 183-235, 1994.

[AFM+03] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems, In *the 1st International Workshop on Formal Modeling and Analysis of Timed Systems*, Marseille, France, September, 2003

[AG04] A. Albert and R. B. GmbH, Comparison of Event-Triggered and Time-Triggered. Concepts with Regard to Distributed Control Systems, In *Proc. of Embedded World*, Nurnberg, Germany, pages 235-252, 2004.

[AGP+99] K. Altisen, G. Gößler, A Pnueli, J. Sifakis, S. Tripakis and S. Yovine, A Framework for Scheduler Synthesis, In *IEEE Real-Time Systems Symposium*, pages 154-163, 1999.

[AM01] Y. Abdeddaïm and O. Maler, Job-Shop Scheduling Using Timed Automata, In *Proc. of 13th International Conference on Computer Aided Verification*, pages 478-492, 2001.

[ÅMH+05] M. Åkerholm, A. Möller, H. Hansson and M. Nolin, Towards a Dependable Component Technology for Embedded System Applications, In *the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORD2005)*, Sedona, Arizona, February 2005.

[AS99] T. F. Abdelzaher and K. G. Shin, Combined Task and Message Scheduling in Distributed Real-Time Systems, *IEEE Transactions on Parallel and Distributed Systems*, volume 10, November 1999.

[ATB93] N. Audsley, K. Tindell and A. Burns, The End of the Line for Static Cyclic Scheduling?, In *Proc. of the 5th Euromicro Workshop Real-Time Systems*, pages 36-41, Oulu, Finland, 1993.

[AV98] J. Axelsson and Volvo technological, A Case Study in Heterogeneous Implementation of Automotive Real-Time Systems, In *the 6th International Workshop on Hardware/Software Codesign*, Seattle, 1998.

[Axe98] J. Axelsson, A Case Study in Heterogeneous Implementation of Automotive Real-Time Systems, In *the 6th International Workshop on Hardware/Software CoDesign*, Seattle, March 15-18, 1998.

[BBD+02] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi, UPPAAL Implementation Secrets, In *Proc. of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and P. Schnoebelen, System and Software Verification: Model-Checking Techniques and Tools, Springer-Verlag, Berlin, Germany, 2001.

[BDL04] G. Behrmann, A. David and K. G. Larsen, A Tutorial on UPPAAL. In *Proc. of the 4th International School on Formal Methods for the Design of Computer*, Communication and Software Systems, LNCC3185, 2004.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, Kronos: a model-checking tool for real-time systems, In *Proc. of the 10th Conference on Computer-Aided Verification*, 1998.

[BHR95] A. Burns, N. Hayes and M. F. Richardson, Generating Feasible Cycle Schedules, In *Control Eng. Practice*, volume 3, no 2, pages 151-162, 1995.

[BHV00] G. Behrmann, T. Hune and F. Vaandrager, Distributed Timed Model Checking – How the Search Order Matters, Technical report, Computing Science Institute, Nijmegen, 2000.

[BJL+98] J. Bengtsson, B. Jonsson, J. Lilius and W. Yi, Partial Order Reductions for Timed Systems, CONCUR'98, pages 485-500, 1998.

[BL92] R. Bettati and J.-S. Liu, End-to-End Scheduling to Meet Deadlines in Distributed Systems, In *Proc. of the IEEE International Conference on Distributed Computing Systems*, pages 452-459, June 1992.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson and P. Pettersson and W. Yi, UPPAAL: A Tool Suit for Validation and Verification of Real-Time System, In *Proc. of Hybrid Systems*, pages 232-243, 1996.

[BLM+98] F. Balarin, L. Lavagno, P. Murthy and A. Sangiovanni-vincentelli, Scheduling for Embedded Real-Time Systems, *IEEE Design & Test*, volume 15, issue 1, pages 71-82, 1998.

[BN03] D. Beyer and A. Noack, Can Decision Diagrams Overcome State Space Explosion in Real-Time Verification?, In *Proc. of the 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems(FORTE),* LNCS 2767, pages 193-208, 2003.

[BNT+93] A. Burns, M. Nicholson, K. Tindell and N. Zhang, Allocation and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System, In *Proc. of the IEEE Workshop on Parallel and Distributed Real-Time System*, California, pages11-20, April 1993.

[Bur91] A. Burns, Scheduling Hard Real-Time Systems: A Review, *Software Engineering Journal*, volume 6, issue 3, pages 116-128, 1991.

[But97] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers, 1997.

[BW01] A. Burns, and A. Wellings, *Real-Time Systems and Programming Languages*, Pearson Education Limited, 2001.

[BW04] J. Bengtsson and W. Yi, Timed Automata: Semantics, Algorithms and Tools, In *Lecture Notes on Concurrency and Petri Nets*, W. Reisig and G. Rozenberg (Editors), LNCS 3098, Springer-Verlag, 2004.

[BY04] Johan Bengtsson and Wang Yi, Timed Automata: Semantics, Algorithms and Tools, In *Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.)*, LNCS 3098, Springer-Verlag, 2004.

[CA95] S. T. Cheng and A. K. Agrawala, Allocation and Scheduling of Real-time Periodic Tasks with Relative Timing Constraints, In *the 2nd International Workshop on Real-Time Computing Systems and Applications*, 1995.

[CE81] E. M. Clarke and E. A. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In *Proc. Logics of Programs Workshop*, Yorktown Heights, New York, volume 131 of Lecture notes in Computer Science, pages 52-71. Springer, May 1981.

[CGP99] E. M. Clarke, Jr, O. Grumberg, and D. A. Peled, Model Checking, The MIT Press, ISBN 0-262-03270-8, 1999.

[DBL+03] A. David, G. Behrmann, K. G. Larsen, and Wang Yi, A Tool Architecture for the Next Generation of UPPAAL, Technical report Uppsala University 2003.

[DP85] R. Dechter and J. Pearl, Generalized best-first search strategies and the optimality af A*, In *Journal of the ACM*, volume 32, pages 505-536, 1985.

[DT98] C. Daws and S. Tripakis, Model Checking of Real-Time Reachability Properties Using Abstractions, TACAS'98, pages 313-329, 1998.

[DY95] C. Daws and S. Yovine, Two Examples of Verification of Multirate timed automata with Kronos, In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pages 66-75, Italy, 1995.

[EH86] E. A. Emerson and J. Y. Halpern, Sometimes and Not Never Revisited: On Branching Versus Linear time Temporal Logic, *Journal of the ACM*, volume 1, pages 151-178, 1986.

[FCB02] J. Fonseca, F. Coutinho and J. Barreiros, Scheduling for a TTCAN network with a stochastic optimization algorithm, In *the 8th International CAN Conference*, Las Vegas, USA, February 2002.

[Feh99] A. Fehnker, Scheduling a Steel Plant with Timed Automata, In *the 6th International Conference on Real-Time Computing Systems and Applications*, IEEE Computer Society Press, 1999.

[FES+98] P. Fancher, R. Ervin, J. Sayer, M. Hagan, S. Bogard, Z. Bareket, M. Mefford and J. Haugen, Intelligent Cruise Control Field Operational Test, Technical Report DOT HS 808 849, Transportation Resarch Institue, The University of Michian, May 1998.

[Fid00] C. J. Fidge, Real-Time Scheduling Theory, The University of Queenland, April 2000.

[FK90] G. Fohler and C. Koza, Heuristic Scheduling for Distributed Hard Real-Time Systems, Technical report, Institut fur Technische Informatik, Technischen Universitat Wien Austria, 1990.

[FMA+00] J. Fonseca, E. Martins, L. Almeida, P. Pedreiras and P. Neves, Flexible Time-Triggered Protocol for CAN – New Scheduling and Dispatching Solutions, In *the 7th International Can Conference*, Amsterdam, 2000.

[Glo89] F. Glover, Tabu Search Part I, ORSA J. on Computing, volume 1, no 3, pages 190-206, 1989.

[Glo90] F. Glover, Tabu Search Part II, ORSA J. on Computing, volume 2, no 1, pages 4-32, 1990.

[Gmb03] R. B. GmbH, ACC Adaptive Cruise Control, ISBN-13: 978-0-8376-1046-7, Bentley Publishers, 2003.

[GR04] J. Goossens and P. Richard, Overview of Real-Time Scheduling Problems, In *Proc. of the 9th international conference on project management and scheduling*, France, April 2004.

[GTW93] F. Glover, E. Taillard and D. Werra, A User's Guide to Tabu Search, Annals of Operations Research, volume 41, pages 3-28, 1993.

[Gur05] R. Gurulingesh, Adaptive Cruise Control, Kanwal Rekhi School of Information Technology Indian Institute of Technology Bombay, 2005.

[Hen02] M. Hendriks, Development of Reactive Programs using UPPAAL, Martijn Hendriks. Department of Computer Science, University of Nijmegen, 2002.

[Hen03] D. Henriksson, Flexible Scheduling Methods and Tools for Real-Time Control Systems, Department of Automatic Control, Lund University, Sweden, 2003.

[HKR98] W. D. Henderson, D. Kendall, A.P. Robson, and S. P. Bradley, An Holistic Approach to Performance Prediction of Distributed Real-Time CAN Systems, In *the 5th International CAN Conference*, San Jose, 1998.

[HLP98] T. Hune and K. G. Larsen and P. Pettersson. Guided Synthesis of Control Programs Using UPPAAL. In *Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation*, pages E15-E22, 1998.

[HNS+92] T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *Proc. of IEEE Symp. On Logic in Computer Science*, 1992.

[Hol75] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, Ann Arbor, MI: University of Michigan Press, 1975.

[HSL+97] K. Havelund, A. Skou, K. G. Larsen and K. Lund, Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL, In *Proc. of the 18th IEEE Real-Time System Symposium*, December 1997.

[Hun01] T. S. Hune, Analyzing Real-Time System: Theory and Tools, Basic Research in Computer Science(BRICS), Department of Computer Science, University of Aarhus, 2001.

[IBA03] International Boxing Association, Rule for International Competitions and Tournaments, http://www.aiba.net, 2003.

[IKL+00] T. Iversen, K, Kristoffersen, K. Larsen, M. Laursen, R. Madsen, S. Mortensen, P. Pettersson and C. Thomasen, Model-Checking Real-Time Control Programs – Verifying LEGO Mindstorms System Using UPPAAL, In *Proc. of the 12th Euromicro Conf. on Real-Time Systems (ECRTS'00)*, 2000.

[JP86] M. Joseph and P. Pandya, Finding response times in a real-time system, *The Computer Journal 29*, volume 5, pages 390-395, 1986.

[Kat99] J. Katoen, Concepts, Algorithms, and Tools for Model Checking, Lehrstuhlfür Informatik, Friedrich-Alexander Universitat Erlängen-Nürnberg, 1999.

[KM85] H. Kopetz and W. Merker, The Architecture of MARS, In *proc. 15th International Symposium on Fault-Tolerant Computing*, pages 274-279, June, 1985.

[Kop91] H. Kopetz, Even-Triggered Versus Time-Triggered Real-Time System, In *Proc. of Int. Workshop on Operating Systems of the 90s and Beyond Lecture Notes in Computer Science*, Berlin, Germany, volume 563, pages 87-101, 1991.

[Kop93] H. Kopetz, Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE TRANS INF. & SYST.*, Vol. E76-D, November 1993.

[Kop95] H. Kopetz, Why Time-Triggered Architectures will Succeed in Large Hard Real-Time Systems, 1995.

[Kop97] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997.

[Kop98] H. Kopetz, A Comparison of CAN and TTP, Technische Universität Wien, Institut für Technische Informatik, 1998.

[Kor85] R. E. Korf, Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, In Artificial Intelligence, volume 27, pages 97-109, 1985.

[LA99] H. Lönn and J. Axelsson, A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications, In *Proc. of the 11th Euromicro Conference on Real-Time Systems*, June 1999.

[LAA+94] G. Liao, E. R. Altman, V. K. Agarwal and G. R. Gao, A comparative Study of MultiProcessor List Scheduling Heuristics, In *Proc. of the 27th Hawaii Int'l Conf. System Sciences*, volume 1, pages 68-77, 1994.

[LH96] K. Lin and A. Herkert, Jitter Control in Time-Triggered Systems, In *Proc. of Hawaii International Conference on System Sciences*, pages 451-459, January 1996.

[LL73] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of ACM*, volume 20, pages 46-61, 1973.

[LP97] H. Lönn and P. Pettersson, Formal Verification of a TDMA Protocol Startup Mechanism, In *Proc. of the pacific Rim International Symposium on Fault-Tolerant Systems*, pages 235-242, December 1997.

[LP00] K. G. Larsen and P. Pettersson, Hybrid & Real-Time Systems in UPPAAL 2K, In MOVEP, June 2000.

[LPY95a] K. G. Larsen, P. Pettersson and W. Yi, Diagnostic Model-Checking for Real-Time Systems, In *Proc. of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, October, 1995.

[LPY95b] K. G. Larsen, P. Pettersson and W. Yi, Model-Checking for Real-Time Systems, In *Proc. of the 10th International Conference on Fundamentals of Computation Theory*, Dresden, Germany, LNCS 965, pages 62-88, August 1995.

[LPY95c] K. G. Larsen, P. Pettersson and W. Yi, Compositional and Symbolic Model Checking of Real-Time Systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pages 76-87, December 1995.

[LPY97] K. G. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, In *Springer International Journal of Software Tools for Technology Transfer 1(1+2)*, volume 1, pages 134-152, October 1997.

[LPY98] M. Lindahl, P. Pettersson and W. Yi, Formal Design and Analysis of a Gearbox Controller, In *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, March 1998.

[MA05] A. Mohammadi and S. G. Akl, Scheduling Algorithms for Real-Time System, Technical Report No. 2005-499, July 2005.

[Mar00] S. Martin, Timing Problems in Distributed Real-Time Computer Control Systems, Mechatronics Lab, Department of Machine Design, Royal Inst. Of Technology, Stockholm, ISSN 1400-1179, 2000.

[MBL+01] M. A. Moncusí, J. M. Banús, J. Labarta and A. Arensa, A New Heuristic Algorithm to Assign Priorities and Resources to Tasks with End-to-End Deadlines, *Parallel and Distributed Techniques and Applications*, CSRA Press, volume1, pages 2102-2108, 2001.

[MFF+01] P. Martí, J. M. Fuertes, G. Fohler and K. Ramamritham, Jitter Compensation for Real-Time Control Systems, In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, London, UK, December 2001.

[MPS95] O. Maler, A. Pnueli and J. Sifakis, On the synthesis of Discrete Controllers for Timed Systems (An Extended Abstract), In *the 12th Annual Symposium on Theoretical Aspects of Computer Science*, Munich, pages 229-242, 1995

[MSM+96] G. Manimaran, M. Shashidhar, A. Manikutty and C. R. Murty, Integrated Scheduling of Tasks and Messages in Distributed Real-Time Systems, Technical report, Dept. of Computer Science and Engg., Indian Institute of Technology, Madras, 1996.

[Nae98] M. Naedele, A Survey of Real-Time Scheduling Tools, November 1998.

[NS00] M. D. Natale and J. A. Stankovic, Scheduling Distributed Real-Time Tasks with Minimum Jitter, *IEEE Transactions on Computers*, volume 49, no 4, April 2000.

[NY00] P. Niebert and S. Yovine, Computing Optimal Operation Schemes for Chemical Plants in Multi-batch Mode. In *Proc. of Hybrid Systems, Computation and Control*, pages 338-351, 2000

[OS97] Y. Oh and S. H. Son, Scheduling Real-Time Tasks for Dependability, *Journal of Operational Research Society*, volume 48, no. 6, pages 629-639, June 1997.

[Pal97] J. C. P. Gutiérrez, J. J. G. García and M. G. Harbour, On the Schedulability Analysis for Distributed Hard Real-Time Systems, In *the 9th Euromicro workshop on Real-Time Systems*, June 1997.

[PEP99a] P. Pop, P. Eles and Z. Peng, Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems, In *the 6th International Conference on Real-Time Computing Systems and Applications*, pages 287-294, December 1999.

[PEP99b] P. Pop, P. Eles and Z. Peng, Communication Scheduling for Time-Triggered Systems, In *the 11th Euromicro Conference on Real-Time Systems*, 1999.

[PEP02] T. Pop, P. Eles and Z. Peng, Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems, In *Proc. of the Tenth International Symposium on Hardware/Software Codesign*, pages 187-192. ACM, 2002.

[Pet99] P. Pettersson, Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice, Ph.D. Thesis, Technical Report, department of Computer Systems, Uppsala University, February 1999.

[PK98] D. T. Pham and D. Karaboga, Intelligent Optimisation Techniques – Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Network, ISBN 1-85233-028-7, Springer-Verlag London, 1998.

[PL00] P. Pettersson and K. G. Larsen, UPPAAL2k, In *Bulletin of the European Association for Theoretical Computer Science*, volume 70, pages 40-44, 2000.

[Pnu77] A. Pnueli, The Temporal Logic of Programs. In *Proc. of the 18th IEEE symp. Foundations of Computer Science*, USA, pages 46-57, 1977.

[Pnu81] A. Pnueli, The Temporal Semantics of Concurrent Programs. Theoretical Computer Science, volume 13, no 1, pages 45-60, 1981

[Pop00] P. Pop, Scheduling and Communication Synthesis for Distributed Real-Time Systems, Ph.D. thesis, Linkopings University, 2000.

[Pop03] T. Pop, Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, Licentiate Thesis no. 1022, Linköping Studies in Science and Technology, 2003.

[PSA97] D. T. Peng, K. G. Shin and T. F. Abdelzaher, Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time System, *IEEE Transactions on Software Engineering*, volume 23, December 1997.

[Rat98] H. H. Rath, XML: Chance and Challenge for Online Information Providers, http://www.xml.org/xml/xml_chance_challenge.shtml, STEP Stürtz Electronic Publishing GmbH, 1998.

[Red98] O. Redell, Global Scheduling in Distributed Real-time Computer Systems an Automatic Control Perspective, Technical report Dept. of Machine design in KTH Royal Institute of technology, Sweden, 1998.

[RFA93] K. Ramamritham, G. Fohler and J. M. Adan, Issues in the Static Allocation and Scheduling of Complex Periodic Tasks, In *the 10th IEEE Workshop on Real-Time Operating Systems and Software*, May 1993.

[RSZ89] K. Ramamritham, J. A. Stankovic and W. Zhao, Distributed Scheduling of Tasks with Deadlines and Resource Requirements, IEEE Transactions on Computers, volume 38, no 8, August 1989.

[SAÅ+04] L. Sha, T. Abdelzaher, K. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky and A. K. Mok, Real Time Scheduling Theory: A Historical Perspective, *Real-Time Systems*, volume 28, pages 101-155, 2004.

[SBM06] R. B. Salah and M. Bozga and O. Maler, On Interleaving in Timed Automata, In *International Conference on Concurrency Theory*, pages 465-476, 2006.

[SPB+00] C. Scheidler, P. Puschner, S. Boutin, E. Fuchs, G. Gruensteidl, Y. Papadopoulos, M. Pisecky, J. Rennhack and U. Virnich, System Engineering of Time-Triggered Architectures – The SETTA Approach, In *Proc. of the 16th IFAC Workshop Distributed Computer Control Systems* (DCCS 2000), Elsevier Science, Amsterdam, 2000.

[Stn92] J. A. Stankovic, Distributed Real-Time Computing: The Next Generation, *Journal of the Society of Instrument and Control Engineers of Japan*, volume 31, pages 726-736, 1992.

[SW97] K. Schild and J. Würtz, Off-line Scheduling of a Real-Time System, In *Proc. of CP97 Workshop on Industrial Constraint-Directed Scheduling*, Schloss Hagenberg, Austria, October. 1997.

[SW98] K. Schild and J. Würtz, Scheduling of Time Triggered Real-Time Systems, Constraints, Kluwer Academic Publishers, volume 5, pages 335-357, 1998.

[TBW91] K. Tindell, A. Burns and A. Wellings, Guaranteeing Hard Real Time End-to-End Communications Deadlines, RTRG/91/107, Department of Computer Science, University of York, 1991.

[TBW92] K. Tindell, A. Burns and A. Wellings, Allocation Hard Real-Time Task (An NP-Hard Problem Made Easy), *Journal of Real-Time Systems*, volume 4, issue 2, pages 145-165, 1992.

[TC94] K. Tindell and A. J. Clark, Holistic Schedulability Analysis for Distributed Hard Real-Time Systems, *Euromicro Journal*, volume 40, pages 117-134, 1994.

[Tha02] N. D. Thai, Real-Time Scheduling in Distributed Systems, *International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)*, pages 165, 2002.

[TLP98] T. Hune and K. G. Larsen and P. Pettersson. Guided Synthesis of Control Programs Using UPPAAL. In *Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation.* Pages E15-E22, 1998.

[Tin00] K. Tindell, Deadline Monotonic Analysis, *Embedded Systems Programming,* volume 13, page 20-38, June 2000.

[UPP02] UPPAAL2k: Small Tutorial, http://www.uppaal.com, 2002

[WNS+05] C. Wilwert, N. Navet, Y. Q. Song and F. Simonot-Lion, Design of Automotive X-by-Wire Systems, In the Industrial Communication Technology Handbook, CRC Press, ISBN 0-8493-3077-7, January 2005.

[WT95] B. Wittenmark and N. Trngren, Timing Problems in Real-Time Control Systems, In *the American Control Conference, Seattle*, Washington, 1995.

[XP93] J. Xu and D. L. Parnas, On Satisfying Timing Constraints in Hard Real-Time Systems, IEEE Transactions on Software Engineering, volume 19, no 1. January 1993.

[XP00] J. Xu and D. L. Parnas, Priority Scheduling Versus Pre-Run-Time Scheduling, In *Int. Journal of Time-Critical Computing System*, volume 18, pages 7-23, 2000.

[Yov96] S. Yovine, Model Checking Timed Automata. In European Educational Forum: School on Embedded Systems, pages 114-152, 1996.

[YPD94] W. Yi, P. Pettersson, and M. Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving, In *Proc. of the 7th Int. Conf. on Formal Description Techniques*, Pages 223-238. North-Holland, 1994.